On Specifying Real-Time Discrete Event Systems : An Application for Designing Real-Time Protocols

A. Khoumsi , G.v. Bochmann and R. Dssouli

Université de Montréal, Faculté des arts et des sciences
Département d'informatique et de recherche opérationnelle
C.P. 6128, Succursale Centre-Ville, Montréal, (Quebec), H3C 3J7

August 1994

ABSTRACT.

This paper deals mainly with modeling and design of distributed communicating systems with temporal requirements. Firstly, timed traces and their corresponding untimed traces are defined and used to model behaviours of real-time discrete event systems (RTDES). These traces use a conceptual digital global clock which generates periodically an event *tick*. Next, a model based on timed automata is defined and studied. This model is convenient to specify a service desired by a user of a distributed RTDES (DRTDES) and the supremal behaviour of the medium. Timed automata use a digital global clock, and several fictitious timers and counters. A second model, based on temporized automata, is used to model the protocol and temporal constraints on the medium. Contrary to timed automata, temporized automata do not use counters. Next, we propose two procedures of protocol synthesis, respectively for sequential and parallel DRTDES. The entries of these procedures are specified with timed automata, while the results of these procedures, i.e., the protocol and the temporal requirements of the medium, are specified with temporized automata. Compared to [10], the application field is much broader, because two important restrictions are removed. Firstly, temporal requirements are between events which are not necessarily consecutive. Secondly, the systems considered can be parallel and concurrent. Compared to [11], three important additions are made. Firstly, the temporized automata are formally defined, and we present the principle to compute them. Secondly, the specifications obtained by the protocol synthesis are optimized in the sense that they do not necessitate to synchronize the different local clocks of each site of the distributed system. Thirdly, the specifications obtained by the protocol synthesis are improved in the sense that they are more concise, by parameterizing some of their transitions.

INDEX TERMS. Concurrent System, Desired Service, Discrete Event System, Protocol Synthesis, Sequential System, Supremal behaviour of the Medium, Temporal Constraint, Temporized Automaton,

1. Introduction

A discrete event system (DES) is a dynamic system where events are executed instantaneously, causing a discrete change of the state of the system. If sequences of events are a regular language, the DES can be specified by a finite automaton. A first example of DES is a telecommunication network; an event can then be the transmission of a packet of data. Another example is a communication protocol, and an event can be execution of a service primitive. For a real-time DES (RTDES), it is not enough to represent the ordering of events, we must also specify temporal constraints on event occurrences. As RTDESs grow in size and complexity, it has become important to develop models and theory which are used to reason about their behaviour. Several models have then been proposed to model and study RTDESs.

1.1. Background literature on modeling real-time discrete event systems

Two approaches have been used to model RTDESs : *Discrete-time* models and *Dense-time* models. Discrete-time models use the domain $\mathbb{N}$ of integers to model time, and some of these models use a fictitious digital global clock which generates a *tick* event [5,24,25]. Time is then viewed as a global state variable that ranges over $\mathbb{N}$, and is incremented by one with every *tick* event. RTDESs are then specified by a timed transition model (TTM), where a fictitious timer T_σ and an interval $[l_\sigma,u_\sigma]$ is associated to each transition σ. As soon as σ becomes firable, T_σ is set to zero and is incremented by one with every *tick* event; σ is enabled only when the value of T_σ belongs to interval $[l_\sigma,u_\sigma]$. RTDESs can also be specified by an untimed transition model (UTM) where the event *tick* is explicitly represented by a transition.

Dense-time models use a dense domain to model time. The latter is then viewed as state variable that ranges over a dense domain and evolves indefinitely. In [1,8,10,31], RTDESs are specified by timed automata (TA), where several clocks are defined. Clocks can be set to zero with the occurrence of any event, and evolve synchronously with time. In [1,8,31], a boolean enabling condition E_σ depending on the value of one or several clocks and a set R_σ of clocks are associated to each transition σ. As soon as σ becomes firable, it may be executed only if E_σ=True. When σ is executed, clocks of R_σ are set to zero. In [10], several enabling conditions $E1_\sigma, E2_\sigma, ..., Ek_\sigma$ are associated to a transition σ, and each of them depends on only one clock. Only one of these enabling conditions is active, depending on the last executed transition. Several other models have been developed, such as time Petri Nets [4,20,21], timed Petri Nets [26], and timed LOTOS [18,19,23].

In this paper, we propose two different models [11] which both use a discrete time. The first model (timed automata) is used to specify the entries of the protocol synthesis (Sect. 4), while the second model (temporized automata) specifies the results of the protocol synthesis (Sect. 6).

1.2. Background literature on protocol design

Two approaches may be used for the design of communication protocols: *Analysis* and *Synthesis* [35]. In the analysis approach [33], the protocol designer starts with an initial version of the protocol, and protocol validation is performed with analysis techniques after the design to detect possible errors and omissions in the design. The sequence of redesign, analysis, error and omission detection, and correction is applied iteratively until the protocol becomes error free.

In the synthesis approach -which is the one we have used-, several methods have been developed [3,6,7,9,10,13,14,15,16,17,27,28,29,30,34,35]. Contrary to analysis, this approach is direct and does not necessitate a validation of the synthesized protocol which is correct by construction. Timing requirements are considered in [10,15,16], but only in particular cases. In [15], the transit delay in the medium is supposed negligible, while in [16] it is bounded by a maximum value. As for [10], timing requirements are only between consecutive events, and the systems considered are sequential. In the present study, these constraints are removed and the systems considered can then be parallel and concurrent. The application field is therefore much broader. Compared to [11], three important additions are made. Firstly, the temporized automata are formally defined (Sect. 6.1) and the principle used to compute them is presented (Sect. 6.2). Secondly, the specifications obtained by the protocol synthesis are optimized in the sense that they do not necessitate to synchronize the different local clocks of each site of the distributed system (Sect. 7 and 8). Thirdly, the specifications obtained by the protocol synthesis are improved in the sense that they are more concise : several transitions are represented by one parameterized transition (Sect. 7.2 and 7.3).
The reasons of using timed, untimed and temporized automata are respectively explained at the beginnings of Sections 4, 5 and 6.

The rest of this paper is organized as follows. In Section 2, we introduce the problem of the protocol derivation. The basic principle used for deriving the protocol is explained. In Section 3, we introduce the models of *timed and untimed traces* used to specify the behaviour of a RTDES. In Section 4, we present the model of *timed automata* used to specify: (a) the service desired by the user; (b) the supremal behaviour of the medium. In Section 5, we present the approach which consists of transforming a timed automaton into an *untimed automaton* containing transitions *tick*. In Section 6, we present the model of *temporized automata* and the approach which consists of transforming an untimed automaton into a temporized automaton. In Sections 7 and 8, we propose two procedures for deriving automatically the specifications of the protocol and of timing constraints on the medium (temporized automata), from the specifications of a desired service and of the supremal behaviour of the medium (timed automata). Section 7 deals with sequential systems, while Section 8 deals with concurrent and parallel systems. And at last, we conclude in Section 9. We will notice that the possible concurrency in the parallel systems, and the timing requirements cause a problem of state space explosion and of complexity.

2. Problem of the protocol synthesis in real-time systems

In a real-time distributed system (RTDS, Fig.1), n protocol entities (with n>1) communicate : **(a)** with the user of the system through several service access points (SAP); **(b)** with each other through a medium assumed reliable. Without a loss of generality, we suppose that to each site i correspond one SAP and one protocol entity, respectively noted SAP_i and PE_i.

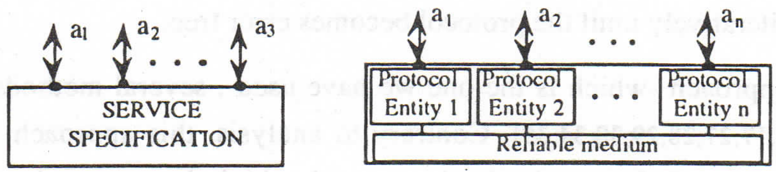


Figure 1. Service and protocol concepts

In the *user's viewpoint*, the RTDS is a black box where only interactions with the user are visible. These interactions correspond to the executions of service primitives (or simply primitives). Therefore, the specification of the service desired by (or provided to) the user defines the *ordering and timing requirements* between the executed primitives.

But in the *designer's viewpoint*, it is necessary to compute the specifications of the local real-time protocol entities PE_i, for i=1,2, ..., n, which may provide the service desired by the user. The designer must also compute timing requirements which must be respected by the medium. In order to avoid the computation of timing requirements impossible to respect by the medium, the designer may refer to a model of a *supremal behaviour* (Sect. 4.2) of the medium, and compute only timing requirements which respect this supremal behaviour. Informally, if for instance we know that the medium needs at least two units of clock time (uct) to carry messages between two protocol entities, this information is contained in the model of the supremal behaviour. In this case, the designer will not compute timing requirements such as : some message must be carried in one uct. We will see that the medium not only carries a message, but it also adds an information about the transit delay of the message in the medium.

The problem of protocol synthesis is then (Fig. 2) to derive systematically the different local protocol specifications and the timing requirements on the medium, from : **(a)** a global specification of the service desired by the user ; **(b)** a model of the supremal behaviour of the medium.

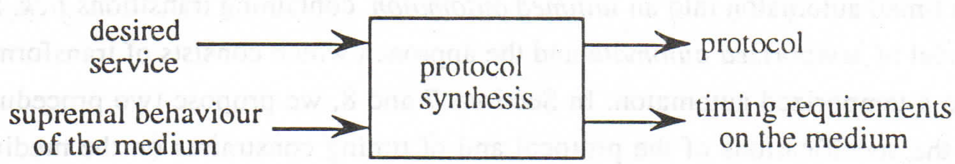


Figure 2. Protocol synthesis

The approach used for deriving protocols is *synthesis* . For the sake of simplicity, we explain the basic principle of protocol synthesis [3,10,13,14,30] only for sequential systems. But parallel systems also are considered, farther in this paper (Section 8). The principle is then : if a primitive A is executed by a protocol entity PE_a, and is followed by execution of a primitive B by PE_b, then after execution of A by PE_a, this one sends a message to PE_b to inform it that it may execute B. If after execution of A by PE_a, there is a choice between several primitives executed by different PE_{bi}, for $i=1,2,\ldots, p$, then PE_a selects one PE_{bi} and sends a message to it to inform it that it may execute one of its primitives. Our main contribution is to consider timing requirements in a more general case than in [10,15,16] (Sect. 1.2).

3. Timed and untimed traces

To model a RTDES, we use a conceptual global digital clock which generates a fictitious event *tick* at a constant frequency; the delay between two consecutive ticks is called *unit of clock time* (uct). The time is then modeled by a global variable noted τ, called *discrete time*, and belonging to the set $\mathbb{N}$ of natural numbers. The variable τ is initially equal to zero and is incremented by one after the passing of each unit of clock time (uct), i.e., after the occurrence of every event *tick* [5,11,24, 25].

3.1. Timed traces and Timed languages

A *finite timed trace* trc over an alphabet V is a finite sequence of pairs $\langle\sigma_i,\tau_i\rangle$, where σ_i is an event of V, and τ_i is an integer such that $\tau_{i+1} \geq \tau_i$. Such trace is represented by trc = $\langle\sigma_1,\tau_1\rangle\ldots\langle\sigma_n,\tau_n\rangle$ and contains all events that have occurred before time τ_n+1. Each $\langle\sigma_i,\tau_i\rangle$ means that the event σ_i has occurred when the discrete time is equal to τ_i. It is clear that there is an inaccuracy of one uct on the exact delay of event occurrences.

An *infinite timed trace* Trc over an alphabet V is an infinite sequence of pairs $\langle\sigma_i,\tau_i\rangle$; any finite prefix of Trc is called a finite timed trace over V. Such infinite trace is represented by Trc= $\langle\sigma_1,\tau_1\rangle\ldots\langle\sigma_i,\tau_i\rangle\ldots$ Each pair $\langle\sigma_i,\tau_i\rangle$ defined in Trc is called a component of Trc which is noted : $\langle\sigma_i,\tau_i\rangle \in$ Trc. Since a τ_i may be equal to τ_{i+1}, several consecutive events may occur at the same discrete time, i.e., during one uct or, in another words, between two ticks of the clock.

Definition 3.1. (Finiteness property)
An infinite timed trace respects the finiteness property (FP) if the number of events executed during one uct is bounded by an arbitrary constant Mc. Formally, Trc = $\langle\sigma_1,\tau_1\rangle\ldots\langle\sigma_i,\tau_i\rangle\ldots$ respects the FP if and only if : $\forall\ i > 0,\ \ \exists\ j > i$ such that $\tau_{j-1} = \tau_i < \tau_j$ and $j \leq i+Mc$. The FP is differently defined in [31], where it only requires that a finite number of events occur in any finite time interval. □

Example 3.1. Let Trc be the following infinite trace Trc=$\langle\sigma_1,2\rangle\langle\sigma_2,4\rangle\ldots\langle\sigma_i,2i\rangle\ldots$ Trc respects the finiteness property because one event occurs when τ is even, and no event occurs when τ is odd. □

Example 3.2. Let Trc be the following infinite trace Trc=$\langle\sigma_1,1\rangle^1\langle\sigma_2,4\rangle^2\ldots\langle\sigma_i,2i\rangle^i\ldots$, where $\langle\sigma,\tau\rangle^p$ means that σ occurs p times when the discrete time is equal to τ. Trc does not respect the FP because the number of events during one uct is not bounded. But Trc respects the FP as it is defined in [31]. □

Definition 3.2. (Timed trace and timed language)
In this paper, we consider only *infinite* timed traces. Such traces, will be simply called *timed traces.* A *timed language* $\mathcal{L}$ over an alphabet V is a set of infinite timed traces over V.
We say that $\mathcal{L}$ respects the finiteness property (FP) if all its timed traces respect the FP. □

Infinite timed traces, which will be simply called *timed traces* , are executed by non terminating processes. This is not really a restriction. In fact, a terminating process which may be executed infinitely often, can also be considered as a non terminating process.

Definition 3.3. (Projection of a timed trace)
Let V be a subset of an alphabet W, and let Trc= $\langle\sigma_1,\tau_1\rangle...\langle\sigma_i,\tau_i\rangle...$ be a timed trace over W. The *projection* of Trc on V, noted $Proj_V(Trc)$, is obtained by removing from Trc all $\langle\sigma_i,\tau_i\rangle$, where $\sigma_i \notin V$. □

Definition 3.4. (Projection and Extension of a timed language)
Let V be a subset of an alphabet W. Let $\mathcal{L}_1$ be a timed language over W. The projection of $\mathcal{L}_1$ on V, noted $Proj_V(\mathcal{L}_1)$, is defined by : $Proj_V(\mathcal{L}_1)=\{Trc, \text{ over } V \mid \exists\ Trce \in \mathcal{L}_1 \text{ with } Trc=Proj_V(Trce)\}$;
Let $\mathcal{L}_2$ be a timed language over V. The extension of $\mathcal{L}_2$ to W, noted $Ext_W(\mathcal{L}_2)$, is defined by :

$$Ext_W(\mathcal{L}_2)=\{Trc \text{ , over } W \mid Proj_V(Trc)\in \mathcal{L}_2\}.$$ □

Remark 3.1. (a) if W=V then $Proj_V(\mathcal{L})=Ext_W(\mathcal{L})=\mathcal{L}$; (b) $Proj_V(Ext_W(\mathcal{L}))=\mathcal{L}$ and $\mathcal{L}\subseteq Ext_W(Proj_V(\mathcal{L}))$.

3.2. Untimed traces and untimed languages

So far, an infinite sequence of events has been represented by a timed trace Trc=$\langle\sigma_1,\tau_1\rangle...\langle\sigma_i,\tau_i\rangle...$
If we represent explicitly the fictitious event *tick* , the same sequence can be represented by an untimed trace TRC= $\alpha_1\alpha_2...\alpha_j$ ..., where each α_j for j=1,2,..., is equal to *tick* or to one of σ_1, σ_2, ...
Example 3.3. The timed Trc=$\langle\sigma_1,2\rangle...\langle\sigma_i,2i\rangle...$ can equivalently be represented by the untimed :
TRC=*tick tick* σ_1 *tick tick* $\sigma_2...\sigma_{i-1}$ *tick tick* σ_i *tick tick* σ_{i+1}... □

A formal definition of the untimed trace corresponding to a timed trace is the following.

Definition 3.5. (Untimed trace, operators *UntimeT* and *TimeT*)
Let Trc = $\langle\sigma_1,\tau_1\rangle...\ \langle\sigma_i,\tau_i\rangle...$ be a timed trace. To obtain the untimed trace TRC corresponding to Trc, we define the operator *UntimeT* , by : TRC=*UntimeT*(Trc)=$\alpha_1\alpha_2 \ldots \alpha_j \ldots$
with : $(\alpha_{i+\tau_i}=\sigma_i)$ and $(\alpha_j=tick$, if $\nexists$ k>0 such that $j=k+\tau_k)$, for i,j=1,2, ...
If Trc respects the finiteness property (Def. 3.1), we also say that TRC respects the finiteness property.
Since the operator *UntimeT* is a bijection, we can define the inverse operator *TimeT*, by :

$$TRC = UntimeT(Trc) \Leftrightarrow Trc=TimeT(TRC)$$ □

Property 3.1. Let TRC= $\alpha_1\alpha_2 \ldots \alpha_j \ldots$ be a infinite untimed trace respecting the finiteness property .
$\exists$ Mc>0 such that : $\forall$ k>0, $\exists$ l_1>k, $\exists$ l_2>k with $\alpha_{l_1} \neq tick$, l_2-k $\leq$Mc+1, and $\alpha_{l_2}=tick$.
Proof : *See Appendix A* □
More informally, Property 3.1 means that the untimed TRC corresponds to an *infinite* timed trace (by $\alpha_{l_1}\neq tick$) which respects the *finiteness property* (by l_2-k $\leq$Mc+1 and $\alpha_{l_2}= tick$).

Definition 3.6. (Untimed language, operators *UntimeL* and *TimeL*)
Let $\mathcal{L}$ be a timed language. $\mathcal{L}^u$, which is called untimed language and noted $\mathcal{L}^u$=*UntimeL*($\mathcal{L}$), is defined by : $\mathcal{L}^u$=*UntimeL*($\mathcal{L}$)={TRC | $\exists$ Trc $\in \mathcal{L}$ with TRC=*UntimeT*(Trc) } □
Since the operator *UntimeL* is a bijection, we can define the inverse operator *TimeL*, by :

$$\mathcal{L}^u = UntimeL(\mathcal{L}) \Leftrightarrow \mathcal{L} = TimeL(\mathcal{L}^u)$$ □

Theorem 3.1. Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two timed languages over a same language.
$UntimeL(\mathcal{L}_1 \cap \mathcal{L}_2) = UntimeL(\mathcal{L}_2) \cap UntimeL(\mathcal{L}_2)$. (**Proof** : *See Appendix A*) □

4. Timed Automata to specify a desired service and a supremal behaviour of the medium

The aim of this section is to define a model based on *timed automata* and used to specify the two entries of the protocol synthesis (Sect. 2), i.e., a global specification of the service desired by the user (Sect. 4.1), and a specification of the supremal behaviour of the medium (Sect. 4.2). Since these two entries are initially unformally specified, timed automata must be as intuitive as possible, so that the transformation from an unformal to a formal specification is easy to determine. Before defining formally the timed automata, let's show in simple examples how timed automata are used to model a global specification of the service desired by the user, and a specification of the supremal behaviour of the medium.

4.1. Service desired by the user

Traditionally, a service desired by the user is defined by the sequences of service primitives which are accepted by the user. But in our case where timing constraints must be respected, two additional kinds of requirements define the desired service; they are the following :

(1) Constraints on the delays between executions of service primitives. In other words, constraints on the numbers of ticks between executions of primitives;

(2) Constraints on the numbers of primitive executions during one unit of clock time. In other words, bounds on the numbers of primitive executions between two ticks of the digital clock.

The service desired by the user of a distributed system is initially unformally specified. Therefore, the model used to formally specify the desired service must be as intuitive as possible, so that the transformation from an unformal to a formal specification is easy to determine. Here is a simple example of a desired service which is initially unformally specified.

Unformal specification :

S_1 : Two service primitives A_1 and B_2 must be executed alternately. The indexes 1 and 2 identify the sites where the primitives are executed, i.e., A_1 and B_2 are respectively executed in sites 1 and 2;

S_2 : Between executions of A_1 and B_2, there may be at most two ticks of the clock;

S_3 : Between executions of B_2 and A_1, there may be at most two ticks of the clock;

S_4 : Between two ticks of the clock, there may be at most the execution of one primitive. Let's notice that S_4 implies the finiteness property (Def. 3.1).

The above unformal specification can be easily formalized as follows.

S_1 can be represented by the two state automaton of Figure 3.a.

S_2 can be formally defined by the use of a fictitious timer t (Def. 4.1):

- the timer is set to zero after the occurrence of A_1,
- a necessary condition of B_2 execution is : $t \leq 2$;

S_3 can be formally defined by the use of the same timer t. In general, when the service is sequential and the timing requirements are between consecutive primitives, then one timer is sufficient :

- the timer is set to zero after the occurrence of B_2,
- a necessary condition of A_1 execution is : $t \leq 2$;

S_4 can be formally defined by the use of a fictitious counter c (Def. 4.3):

- the counter c is set to zero after every tick,
- c is incremented by one after execution of every primitive,
- a necessary condition of every primitive execution is : $c < 1$.

Such specification can be formally represented by the timed automaton (TA) of Figure 3.b. Each transition of the TA is then defined by :

- starting and reached states;
- two enabling conditions ($t \leq 2$ and $c < 1$);
- a set of timers which are set to zero with the occurrence of the transition ({t}).

3.a. Automaton 3.b. Timed automaton

Figure 3. Example of service specification

4.2. Supremal behaviour of the medium

The specification of the supremal behaviour of the medium is used in order to avoid, in the protocol synthesis, the derivation of timing requirements impossible to respect by the medium (Sect. 2) which is assumed reliable. The supremal behaviour is defined by a timed automaton $SupMed^t_{i,j}$ (Fig. 4) for every oriented pair of sites i and j, where site i is the sender and site j is the receiver. Let then the events s^j_i and r^i_j meaning respectively "Site i sends a message to site j" and "Site j receives a message coming from site i". $SupMed^t_{i,j}$ specifies that the number of ticks between s^j_i and r^i_j belongs to a given interval $I_{i,j}=[t^{min}_{i,j}; t^{max}_{i,j}]$, where $t^{min}_{i,j}$ and $t^{max}_{i,j}$ are constant integers such that $1 \leq t^{min}_{i,j} \leq t^{max}_{i,j} < \infty$. Therefore, we suppose that there is at least one tick of the clock during the transmission of a message.

Temporal constraints between s^j_i and r^i_j can be formally defined by the use a fictitious timer $t_{i,j}$:

- $t_{i,j}$ is set to zero after the occurrence of s^j_i ,
- a necessary condition of r^i_j occurrence is : $E_{i,j}(t_{i,j})=(t_{i,j} > (t^{min}_{i,j} -1)) \wedge (t_{i,j} \leq t^{max}_{i,j})$.

Contrary to example in Section 4.1, no counter is used because timing requirements ensure the finiteness property (Def.3.1) due to $1 \leq t^{min}_{i,j}$. Therefore the second enabling condition of transitions in $SupMed^t_{i,j}$ is always True. The timed automaton $SupMed^t_{i,j}$ is represented on Figure 4.

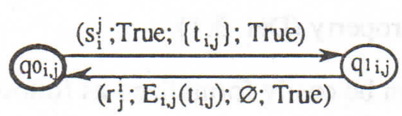


Figure 4. Supremal behaviour $SupMed^t_{i,j}$ of the medium for a pair (sender i, receiver j)

Remark 4.1. Timing requirements on $SupMed^t_{i,j}$ and $SupMed^t_{j,i}$ may be different.

In the remaining part of the present section 4, timed automata are formally defined and studied.

For defining a TA, which is an extended FSM accepting a timed language (Def. 3.2), we use in general:

- a global digital clock which generates the event tick, and then informs about the passing of time,
- a finite set of fictitious digital timers (Def. 4.1), for specifying the timing requirements,
- a finite set of counters (Def. 4.3), for respecting properties stronger than the finiteness property (Def.3.1).

4.3. Timers and counters

Definition 4.1. (Timer and timer state)

A fictitious timer t_i is a variable which belongs to the set $\mathbb{N}$ of natural numbers. t_i is automatically incremented by one with every tick. The operations we can do on t_i are :

- Reset : a timer t_i, which is increasing regularly with every tick, can be set to zero. Therefore, t_i represents the time elapsed from the last reset .
- Comparison : t_i can be compared to a constant integer. The comparison operators are $=$, $>$ and $\leq$.

Other operators $<$ and $\geq$ are not necessary because timer values are integers.

Initially, when the discrete time τ is equal to zero, t_i also is equal to zero.

Let the Nt-uplet $ts=(t_1,..., t_{Nt})$, where Nt (or |T|) is the number of timers $t_1, t_2, ..., t_{Nt}$. Any value of ts is called *timer state* . □

We deduce that if several timers $t_1, t_2, ..., t_{Nt}$ are used, then they are automatically and *simultaneously* incremented with every tick, i.e., when the discrete time τ is incremented. Therefore, all the timers are synchronized on the digital global clock.

Definition 4.2. (T_Condition, set E_T)

Let $T=\{t_1, t_2, ..., t_{Nt}\}$ be a set of timers. A T_Condition E(ts), w.r.t. T, is a boolean function which associates to a timer state a value TRUE of FALSE. E(ts) is formed from :

(a) canonical boolean functions $t_i \sim k$, where $k \in \mathbb{N}^*$, and $\sim$ is $=$, $\leq$ or $>$;

(b) operators AND($\wedge$), OR($\vee$), and NOT($\neg$) on these canonical boolean functions.

The set of all T_Conditions, w.r.t. T, is noted E_T. □

Definition 4.3. (Counter and counter state)

A fictitious counter c_i, w.r.t. an alphabet Vc_i, is a variable belonging to $\mathbb{N}$. c_i is automatically :

(a) incremented after the occurrence of any event of Vc_i;

(b) set to zero with every tick, i.e.,when τ is incremented.

Let the Nc-uplet $cs=(c_1,..., c_{Nt})$, where Nc (or |C|) is the number of counters $c_1, c_2, ..., c_{Nt}$. Any value of cs is is called *counter state* . □

Definition 4.4. (F_Condition, set E_C)

Let $C=\{ c_1, c_2, ..., c_{Nc}\}$ be a set of counters. A F_Condition K(cs), w.r.t. C, is a boolean function which associates to a counter state a value TRUE or FALSE. K(cs) is formed from :

(a) canonical boolean functions $c_i < Mc_i$, where $Mc_i \in \mathbb{N}^*$;

(b) operator AND($\wedge$) on these canonical functions. The set of all F_Conditions, w.r.t. C, is noted E_C. □

4.4. Timed automata

Let $A=(Q,V,\delta,q_0)$ be a FSM where Q is a set of states, V is an alphabet, q_0 is the initial state, and $\delta \subseteq Q \times V \times Q$ defines the transitions, i.e., a transition of A can be represented by $[q_1;\sigma;q_2]$. Let's see how a timed automaton can defined from the FSM A.

Definition 4.5. (Enabled and eligible timed transition, Reset)
Let $T=\{t_1, ..., t_{Nt}\}$ be a set of timers, and let $C=\{c_1, ..., c_{Nc}\}$ be a set of counters, w.r.t $Vc_i \subseteq V$, for $i=1,2, ..., N_C$. Let E_T (resp. E_C) be the sets of T_Conditions (resp. F_Conditions), w.r.t. T (resp. C).
A *timed transition* , w.r.t. A and T and C, is defined by $Tr=[q_1;\sigma;q_2;E(ts);R;K(cs)]$, where $[q_1;\sigma;q_2] \in \delta$, $E(ts) \in E_T$, $K(cs) \in E_C$, and $R \subseteq T$. R is called *Reset* of the transition Tr. The semantics of Tr is the following. Let q_1 be the current state :
(1) σ may occur only if E(ts) (Def.4.2).*and* K(cs) (Def.4.4) are true;
(2) after the occurrence of σ : (a) the state q_2 is reached, timers of R are set to zero, *and*
(b) c_i is incremented if $\sigma \in Vc_i$, for $i=1,2, ..., N_C$.
Besides, $K(cs)=(c_{i1}<Mc_{i1})\wedge...\wedge(c_{ip}<Mc_{ip})$, where $c_{i1},..., c_{ip}$ are all counters respectively w.r.t. $Vq_1,..., Vq_{ip}$, such that $\sigma \in Vc_{i1} \cap ... \cap Vc_{ip}$.
A timed transition $[q_1;\sigma;q_2;E(ts);R;K(cs)]$ is *enabled* if : q_1 is the current state and $E(ts) \wedge K(cs))$ is true.
A timed transition $Tr=[q_1;\sigma;q_2;E(ts);R;K(cs)]$ is *eligible* if :

Tr is enabled or will become enabled with the passing of time (without occurrence of any event). □

A timed automaton (TA) A^t can then be constructed if we transform each transition $tr=[q_1;\sigma;q_2]$ of A into a timed transition Tr by associating to it, a T_condition E(ts), a Reset, and a F_Condition K(cs).

Definition 4.6. (Timed automaton)
Formally, a timed automaton $A^t=(Q,V,T,\mathcal{V},\delta,q_0)$ is defined as follows. Q is the set of states, q_0 is the initial state, V is the alphabet, T is the set of timers $t_1, t_2,...,t_{Nt}$, $\mathcal{V}=\{Vc_i \mid \text{for } i=1,2,...,Nc\} \subset 2^V$, where each Vc_i is associated to one counter c_i . $\delta \subseteq Q \times V \times Q \times E_T \times 2^T \times E_C$ defines the timed transitions, where E_T and E_C are the sets of T_Conditions and F_Conditions (Def.4.2 and 4.4). Besides, A^t *accepts only infinite timed traces* (Def. 4.10), and is called a TA. □

Remark 4.2. In the particular case where no timer (resp. counter) is used, then the T_Conditions (resp. F_Conditions) of all transitions are equal to True. If Nc=0, then $\mathcal{V}=\emptyset$. □

Example 4.1. Let's consider a communicating system which executes the three following service primitives : connect.request, connect.confirm, and disconnect.indication. These primitives are respectively abbreviated by *cr* , *cc* , *di* . The informal desired behaviour is the following. The primitive *cr* is first executed. It can be accepted and followed by *cc* , or refused and followed by *di* . And this process is repeated indefinitely. Between two consecutive *cr* , there may be at most 9 ticks. After *cc* or *di*, we must wait at least 3 ticks before the next *cr*. After its execution, if *cr* is not refused (i.e., not followed by *di*) 2 ticks after its occurence, it will be inevitably accepted (i.e., followed by *cc*) within 3 ticks after its occurence. With this informal specification, the finiteness property (Def. 3.1) is automatically respected, because of the minimum 3 ticks between *cc* or *di* and *cr*.

This desired behaviour is formally specified by the TA of Figure 5, which uses two timers t_1 and t_2. t_1 is used for defining timing requirements between : two *cr*, *cr* and *cc*, *cr* and *di*. t_2 is used for defining timing requirements between : *cc* and *cr*, *di* and *cr*. In this example, the use of counters is not mandatory, because the timing requirements ensure the finiteness property. But in the general case, where timing requirements do not ensure the finiteness property, at least one counter must be used. In this example, Nt=2, $T=\{t_1,t_2\}$, $ts=(t_1,t_2)$, Nc=0 and $C=\varnothing$. Therefore, the F_Condition of all timed transitions is True (Remark 4.2). The T_Conditions are $E_1(ts)=((t_1\leq 9)\wedge(t_2>2))$, $E_2(ts)=(t_1\leq 3)$, $E_3(ts)=(t_1\leq 2)$, and the Resets are $R_1=\{t_1\}$, $R_2=\{t_2\}$, $R_3=\{t_2\}$. The TA of Figure 5 is then defined by $A^t=(Q,V,T,\varnothing,\delta,q_0)$ where: $Q=\{q_0,q_1\}$, $V=\{cr,cc,di\}$, $T=\{t_1,t_2\}$, $C=\varnothing$, $\delta=\{[q_0;cr;q_1;E_1;R_1;\text{True}], [q_1;cc;q_0;E_2;R_2;\text{True}], [q_1;di;q_0;E_3;R_3;\text{True}]\}$. □

Let's mention that a timed transition $Tr=[q;\sigma;r;E;R;K]$ is represented graphically by : $q \xrightarrow{(\sigma;E;R;K)} r$

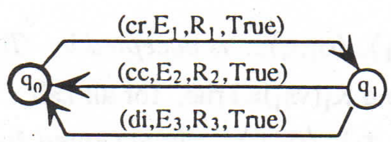


Figure 5. Timed automaton

As for example of Section 4.2, $SupMed^t_{i,j}$ is formally defined by $(Q_{i,j},V_{i,j},\{t_{i,j}\},\{V_{i,j}\},\delta_{i,j},q0_{i,j})$, where $Q_{i,j}=\{q0_{i,j},q1_{i,j}\}$, $V_{i,j}=\{s^j_i,r^i_j\}$, $\delta_{i,j}=\{[q0_{i,j},s^j_i,q1_{i,j},\text{True},\{t_{i,j}\},\text{True}], [q1_{i,j},r^i_j,q0_{i,j},E_{i,j}(t_{i,j}),\varnothing,\text{True}]\}$, with : $E_{i,j}(t_{i,j})=(t_{i,j}>(t^{min}_{i,j}-1))\wedge(t_{i,j}\leq t^{max}_{i,j})$.

Definition 4.7. (set $\mathcal{T}$ of timer states)

Let $T=\{t_1, ..., t_{Nt}\}$ be a set of timers used for defining a TA A^t, and let Mt_i be the maximum value a timer t_i is compared to, for defining the T_Conditions (Def.4.2) of all the transitions of A^t. In this case, t_i does not need to be incremented as soon as $t_i=Mt_i+1$. In fact, in this case the incrementation would have no influence on truths of the T_Conditions. Therefore, we can limit t_i by Mt_i+1, for i=1, 2, ..., Nt, and the set $\mathcal{T}$ of timer states $ts=(t_1, ..., t_{Nt})$ is equal to or included in $\langle 0;Mt_1+1\rangle\times...\times\langle 0;Mt_{Nt}+1\rangle$, where $\langle 0;Mt_i+1\rangle$ is the set of integers belonging to the interval $[0;Mt_i+1]$. □

In Example 4.1, $Mt_1=9$, and $Mt_2=2$, and then $\mathcal{T}\subseteq\langle 0;10\rangle\times\langle 0;3\rangle$

Definition 4.8. (Addition between $\mathcal{T}$ and $\mathbb{N}$)

Let $T=\{t_1, ..., t_{Nt}\}$ be a set of timers used for defining a TA A^t. The addition between $\mathcal{T}$ and $\mathbb{N}$ is defined as follows : if $ts=(t_1, ..., t_{Nt})\in\mathcal{T}$ and $p\in\mathbb{N}$, then $ts+p=(\inf(t_1+p,Mt_1+1),...., \inf(t_{Nt}+p, Mt_{Nt}+1))$. Where inf is defined by : $\inf(A,B)\in\{A,B\}$ and $((\inf(A,B)=A)\Leftrightarrow(A\leq B))$. □

Intuitively, if ts is the current timer state, then ts+p is the futur timer state after the occurrences of p ticks of the clock. In Example 4.1, if ts=(4,1) and p=3, then $ts+3=(\inf(4+3;10), \inf(1+3;3))=(7,3)\neq(7,4)$.

Definition 4.9. (set C of counter states)

Let $C=\{c_1, ..., c_{Nc}\}$ be a set of counters used for defining a TA A^t, and let Mc_i be the maximum value which bounds c_i. Therefore, the set C of counter states $cs=(c_1, ..., c_{Nc})$ is equal to or included in $\langle 0;Mc_1\rangle \times ... \times \langle 0;Mc_{Nc}\rangle$, where $\langle 0;Mc_i\rangle$ is the set of integers belonging to the interval $[0;Mc_i]$. □

In example of Section 4.1, $Mc_1=1$, and then $C \subseteq \langle 0;1\rangle$.

Definition 4.10. (Acceptance of a timed trace and of a language, equivalence, partial order relation)

Let A^t be a TA $(Q,V,T,\mathcal{V},\delta,q_0)$, with $T=\{t_1, ..., t_{Nt}\}$, $\mathcal{V}=\{Vc_1,...,Vc_{Nc}\}$, and then $C=\{c_1, ..., c_{Nc}\}$.

Let $\mathcal{Tr}=Tr_1 Tr_2 ... Tr_i ...$ be an infinite sequence of transitions of A^t, with :

$$\forall i \in \mathbb{N}^* : \; Tr_i=[q_{i-1};\sigma_i;q_i;E_i(ts);R_i;K_i(cs)] \in \delta.$$

Let $\mathcal{R}_i$ be a function which sets to zero all timers in $R_i \subseteq T$, i.e., $\mathcal{R}_i(t_1,...,t_{Nt})=(x_1,...,x_{Nt})$ where :

$x_j=0$ if $t_j \in R_i$, and $x_j=t_j$ if $t_j \notin R_i$, for $j=1, ..., Nt$.

Let $\mathcal{S}_i$ be a function which updates cs with the occurrence of event σ_i, i.e., $\mathcal{S}_i(c_1,...,c_{Nc})=(y_1,...,y_{Nc})$

where : $y_j=c_j+1$ if $\sigma_i \in Vc_j$, and $y_j=c_j$ if $\sigma_i \notin Vc_j$, for $j=1, ..., Nc$.

Let : - $\tau_0=0$, the Nt-uplet $ts_0=(0,...,0)$, and the Nc-uplet $cs_0=(0,...,0)$

- For all $i>0$: $us_i=ts_{i-1}+\tau_i-\tau_{i-1}$ and $ts_i=\mathcal{R}_i(us_i)$; $vs_i=0$ if $\tau_i>\tau_{i-1}$, $vs_i=cs_{i-1}$ if $\tau_i=\tau_{i-1}$, and $cs_i=\mathcal{S}_i(vs_i)$

- The infinite timed trace $Trc=\langle\sigma_1,\tau_1\rangle...\langle\sigma_i,\tau_i\rangle...$ *is accepted by* $\mathcal{Tr}$, if and only if :

$$E_i(us_i)=\text{True and } K_i(vs_i)=\text{True, for all } i>0.$$

- The infinite timed trace $Trc=\langle\sigma_1,\tau_1\rangle...\langle\sigma_i,\tau_i\rangle...$ *is accepted by* A^t, if and only if there exists an infinite sequence $\mathcal{Tr}$ of transitions of A^t which accepts Trc.

Informally, a system specified by A^t may execute a trace accepted by A^t.

- A timed language, noted $\mathcal{L}_{A^t}$, *is accepted by* A^t if it contains all and only the traces accepted by A^t.
- A^t_1 and A^t_2 are *equivalent* , and noted $A^t_1 \equiv A^t_2$, if and only if $\mathcal{L}_{A^t_1}=\mathcal{L}_{A^t_2}$.
- A^t_1 *is smaller than or equal to* A^t_2, and noted $A^t_1 \leq A^t_2$, if and only if $\mathcal{L}_{A^t_1} \subseteq \mathcal{L}_{A^t_2}$. □

Property 4.1. Let $A^t=(Q,V,T,\mathcal{V},\delta,q_0)$ be a timed automaton specifying a non terminating system, with $\mathcal{V}=\{Vc_1, Vc_2,...,Vc_{Nc}\} \subseteq 2^V$. If $Vc_1 \cup ... \cup Vc_{Nc}=V$, then the language $\mathcal{L}_{A^t}$ accepted by A^t (Def. 4.10) respects the finiteness property. In this case, we say that A^t respects the finiteness property.

Proof : *See Appendix A* . □

4.5. Product of timed automata

The global desired service may be made up of several services concurrent with each other (Def. 4.11). If every of these services is specified by a TA, we show in the present section how to compute the TA which specifies the global service. For the sake of simplicity and without a loss of generality, we consider only the case where there are two concurrent services.

Definition 4.11. (Independent and concurrent DES)

Let A^t_i be two TA over alphabets Vi, for i=1, 2, specifying two processes.

If $V1 \cap V2=\emptyset$, the two processes are independent with each other.

If $V1 \cap V2 \neq \emptyset$, the two processes are concurrent. In fact, they may run in parallel by executing respectively events of $V1 - V2$ and $V2 - V1$, but they must execute conjointly events of $V1 \cap V2$. □

4.5.1. Product of two timed automata over the same alphabet

Let A_1^t and A_2^t be two TAs (Def. 4.6) defined over the same alphabet V. An intuitive definition of the synchronized product of A_1^t and A_2^t, noted $A_1^t \times A_2^t$, is the following : $A_1^t \times A_2^t$ is a TA specifying a system which may execute *all and only* the infinite timed traces accepted by both A_1^t and A_2^t.

Definition 4.12. (Product over a same alphabet)
Let $A_i^t=(Qi,V,Ti,\mathcal{V}i,\delta i,qi_0)$, for $i=1,2$, be two TA over a same alphabet V, with $T1 \cap T2=\emptyset$, and $\mathcal{V}i=\{Vci_1,...,Vci_{Nci}\}$. Each A_i^t uses then a set $Ti=\{ti_1,...,ti_{Nti}\}$ of timers and a set $Ci=\{ci_1,...,ci_{Nci}\}$ of counters, where each ci_j is w.r.t. Vci_j. The product, noted $A^t=A_1^t \times A_2^t$, is defined by $A^t=(Q,V,T,\mathcal{V},\delta,q_0)$, with $\mathcal{V}=\mathcal{V}1 \cup \mathcal{V}2$, $T=T1 \cup T2$, $Q \subseteq Q1 \times Q2$, $q_0=\langle q1_0,q2_0 \rangle \in Q$, and :
<u>Definition of δ</u> : Let E_{T1}, E_{T2} and E_T be the set of T_Conditions (Def. 4.2), respectively w.r.t. T1, T2 and $T=T1 \cup T2$. Let E_{C1}, E_{C2} and E_C be the set of F_Conditions (Def. 4.4), respectively w.r.t. C1, C2 and $C=C1 \cup C2$. Then $\forall \langle q1,q2 \rangle, \langle r1,r2 \rangle \in Q$, $\forall \sigma \in V$, $\forall E \in E_T$, $\forall R \subseteq T$, $\forall K \in E_C$:

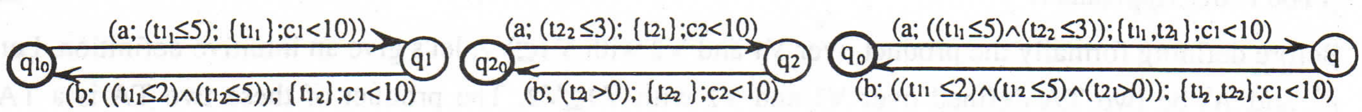
$$([\langle q1,q2\rangle,\sigma,\langle r1,r2\rangle,E,R,K] \in \delta) \Leftrightarrow (\exists E1 \in E_{T1}, \exists E2 \in E_{T2}, \exists R1 \subseteq T1, \exists R2 \subseteq T2, \exists K1 \in E_{C1}, \exists K2 \in E_{C2},)$$
$$(\text{with}: R=R1 \cup R2, \quad E=E1 \wedge E2, \quad K=K1 \wedge K2, \quad \text{and} \quad)$$
$$([q1,\sigma,r1,E1,R1,K1] \in \delta 1, \quad \text{and} \quad [q2,\sigma,r2,E2,R2,K2] \in \delta 2. \quad) \qquad \square$$

Theorem 4.1. If $\mathcal{L}_{A_1^t}$ and $\mathcal{L}_{A_2^t}$ are respectively the timed languages accepted by A_1^t and A_2^t over the same alphabet, then: $\mathcal{L}_{A_1^t \times A_2^t} = \mathcal{L}_{A_1^t} \cap \mathcal{L}_{A_2^t}$. **(Proof :** *See Appendix A* **).** $\square$

Property 4.2. In Def. 4.12, if $Vc1_1 \cup ... \cup Vc1_{Nc1} = Vc2_1 \cup ... \cup Vc2_{Nc2} = V$, then A_1^t, A_2^t, and $A_1^t \times A_2^t$ respect the finiteness property. **(Proof :** *See Appendix A* **).** $\square$

Remark 4.3 : (a) In Def.4.12, if there exist $i \leq Nc_i$ and $j \leq Nc_j$ such that $Vc1_i=Vc2_j$, then counters $c1_i$ and $c2_j$ are equal, because they are incremented and set to zero simultaneously. Therefore, only one of them, for example $c1_i$, is used to define A_1^t, A_2^t, and $A_1^t \times A_2^t$.
(b) From Theorem 4.1, we deduce that if A_1^t and A_2^t specify two sequential processes over the same alphabet, then *their synchronized product also specifies a sequential process.*

Example 4.2. A_1^t and A_2^t are respectively represented on Figures 6.a and 6.b. $A_1^t=(Q1,V,T1,\mathcal{V},\delta 1,q1_0)$ and $A_2^t=(Q2,V,T2,\mathcal{V},\delta 2,q2_0)$, with $\mathcal{V}=\{Vc1\}=\{Vc2\}=\{V\}$, $Q1=\{q1_0,q1\}$, $T1=\{t1_1,t1_2\}$, $Q2=\{q2_0,q2\}$, $T2=\{t2_1,t2_2\}$, $V=\{a,b\}$, and $Mc1=Mc2=10$. $\delta 1=\{[q1_0,a,q1,E1_1,\{t1_1\},K1], [q1,b,q1_0,E1_2,\{t1_2\},K1]\}$, with: $E1_1=(t1_1 \leq 5)$, $E1_2=(t1_1 \leq 2) \wedge (t1_2 \leq 5)$, and $K1=(c1<10)$. $\delta 2=\{[q2_0,a,q2,E2_1,\{t2_1\},K2], [q2,b,q2_0,E2_2,\{t2_2\},K2]\}$, with: $E2_1=(t2_2 \leq 3)$, $E2_2=(t2_1>0)$, and $K2=(c2<10)$. Since $V=Vc1=Vc2$, only one counter, for example c1, is used (Remark 4.3.a), and transitions of A_1^t, A_2^t, and $A_1^t \times A_2^t$ are enabled only if $(c1<10)$. The synchronized product of A_1^t and A_2^t is represented on Figure 6.c. $\square$



6.a. A_1^t 6.b. A_2^t 6.c. $A^t = A_1^t \times A_2^t$

Figure 6. Synchronized product over the same alphabet

4.5.2. Product of two timed automata over alphabets V_1 and V_2 with $V_1 \subseteq V_2$

Before defining the product over alphabets V_1 and V_2, with $V_1 \subseteq V_2$, let's give two definitions.

Definition 4.13. (Operator $\oplus$ on E_T)

Let $E_1(ts)$, $E_2(ts)$, ..., $E_k(ts)$ be k T_Conditions (Def. 4.2), depending on a set of timers $\{t_1, t_2, ..., t_{Nt}\}$. We define $E(ts)=E_1(ts)\oplus E_2(ts)\oplus...\oplus E_k(ts)$ as follows.

$$(E(ts)=False) \Leftrightarrow \{\forall i \in \{1,...,k\}, \forall p \in \mathbb{N}: E_i(ts+p)=False\}$$ □

Informally, $E_1(ts)\oplus...\oplus E_k(ts)$ is false if and only if all $E_i(ts)$ are false and remain false with the passing of time. If for instance $T=\{t_1\}$, $E_1(t_1)=(t_1\leq 5)$, $E_2(t_1)=((t_1>2)\wedge(t_1\leq 6))$, then $E(t_1)=E_1(t_1)\oplus E_2(t_1)=(t_1\leq 6)$.

Definition 4.14. (Extension of a timed automaton)

Let $A^t=(Q,V,T,\mathcal{V},\delta,q_0)$ be a TA over an alphabet V with $T=\{t_1,...,t_{Nt}\}$, $\mathcal{V}=\{Vc_1,...,Vc_{Nc}\}$, and then $C=\{c_1,...,c_{Nc}\}$. Let E_T (resp. E_C) be the set of T_Conditions w.r.t. T (resp. F_Conditions w.r.t. C). Let W be an alphabet such that $V\subseteq W$. The *extension* of A^t to the alphabet W, noted $Ext_W(A^t)$, is a TA defined by $(Q,W,T,\mathcal{V},\delta_{ext},q_0)$, where $\delta_{ext}\subseteq Q\times W\times Q\times E_T\times 2^T\times E_C$ is such that :

(1) $\forall$ $q1,q2\in Q$, $\forall$ $\sigma\in V$, $\forall$ $E\in E_T$, $\forall$ $R\subseteq T$, $\forall$ $K\in E_C$: $[\langle q1,\sigma,q2,E,R,K\rangle \in \delta \Leftrightarrow [\langle q1,\sigma,q2,E,R,K\rangle \in \delta_{ext}$.

(2) $\forall$ $q\in Q$: Let $E_i \in E_T$, for i=1,..., k, be all the T_Conditions of E_T such that : $\exists$ $q_i\in Q$, $\exists$ $\sigma_i\in V$, $\exists$ $R_i\in 2^T$, $\exists$ $K_i\in E_C$, with $[\langle q,\sigma_i,q_i,E_i,R_i,K_i\rangle \in \delta$, and let then $E= E_1\oplus E_2\oplus ... \oplus E_k$.

Then $\forall\sigma\in W\text{-}V$: $[\langle q,\sigma,q',E',R,K\rangle\in \delta_{ext} \Leftrightarrow (q'=q, E'=E, R=\emptyset, K=True)$.

If $B^t=Ext_W(A^t)$, then A^t is called projection of B^t in the alphabet V, and is noted $A^t=Proj_V(B^t)$. □

Informally, $Ext_W(A^t)$ is obtained by adding selfloops of all events of W-V to each state of A^t. The resets of these selfloops are empty, and their T_conditions are defined as follows. The T_Condition of the added selfloops at a state q of A^t is true if at least one of the transitions defined in A^t from q is eligible (Def. 4.5.) The F_Condition for events of W-V is always true, and then $Ext_W(A^t)$ does not necessarily respect the finiteness property (Property 3.1).

Intuitively, let $\mathcal{P}_{ext}$ and $\mathcal{P}$ be two non terminating processes respectively specified by $Ext_W(A^t)$ and A^t, where A^t is defined over the alphabet V. An external agent who can observe all and only the events of V, cannot differentiate the two processes. If the T_Conditions of the added selfloops in $Ext_W(A^t)$ were always true, the external agent may see $\mathcal{P}_{ext}$ as a terminating process. In fact in this case, it is possible that a selfloop of an event of W-V is indefinitely executed. In Example 4.3 (next Section 4.5.3), the two timed automata of Figures 7.a. and 7.b. are extended into the two timed automata of Figures 8.a and 8.b.

Lemme 4.1. If $\mathcal{L}_{A^t}$ is the timed language accepted by a TA A^t over an alphabet V, and if W is an alphabet such that $V\subseteq W$, then: $\mathcal{L}_{Ext_W(A^t)}=Ext_W(\mathcal{L}_{A^t})$. (see Def.3.4 for $Ext_W(\mathcal{L}_{A^t})$)

(Proo f : *See Appendix A* **).** □

Before defining formally the product over V1 and V2 with $V_1\subseteq V_2$, let's give an intuitive definition. Let A_1^t and A_2^t be two TA defined over V1 and V2 with $V_1\subseteq V_2$. The product of these two TA is a TA specifying a system which may execute *all and only* the infinite timed traces which both :

are accepted by A_2^t, and whose projections (Def. 3.3) on V1 are accepted by A_1^t .

Definition 4.15. (Product over V1 and V2 with $V1 \subseteq V2$)

Let $A_i^t=(Qi,Vi,Ti,\mathcal{V}i,\delta i,qi_0)$, for i=1,2, be two TA (Def. 4.6) over alphabets V1 and V2, with $V1 \subseteq V2$, $T1 \cap T2=\emptyset$, and $\mathcal{V}i=\{Vci_1,...,Vci_{Nci}\}$, i.e., each A_i^t uses a set $Ci=\{ci_1,...,ci_{Nci}\}$ of counters where each ci_j is w.r.t. Vci_j. Their synchronized product, noted $A_1^t \otimes A_2^t$, is defined by :

$A_1^t \otimes A_2^t=(Q,V2,T1 \cup T2,\mathcal{V}1 \cup \mathcal{V}2,\delta,q_0)=Ext_{V2}(A_1^t) \times A_2^t$ (See Def.4.12 and 4.14 for $\times$ and $Ext_{V2}(A_1^t)$). □

Theorem 4.2. If $\mathcal{L}_{A_1^t}$ and $\mathcal{L}_{A_2^t}$ are respectively the timed languages accepted by A_1^t and A_2^t respectively over alphabets V1 and V2, with $V1 \subseteq V2$, then: $\mathcal{L}_{A_1^t \otimes A_2^t}=\mathcal{L}_{Ext_{V2}(A_1^t)} \cap \mathcal{L}_{A_2^t}$. **(Proof :** *See Appendix A*). □

Property 4.3. Let A_1^t and A_2^t be two TA, respectively over alphabets V1 and V2 with $V1 \subseteq V2$. If A_2^t respects the finiteness property (FP), then $A_1^t \otimes A_2^t$ respects the FP. **(Proof :** *See Appendix A*). □

Remark 4.4. (a) in Definition 4.15, if V1=V2, then $A_1^t \otimes A_2^t=A_1^t \times A_2^t$ (Def. 4.12), because $Ext_{V2}(A_1^t)=A_1^t$; (b) From Theorem 4.2, we deduce that if A_1^t and A_2^t specify two sequential processes respectively over alphabets V1 and V2 with $V1 \subseteq V2$, then *their synchronized product also specifies a sequential process.*

4.5.3. General parallel product of two timed automata

Before defining formally the parallel product of two TA A_1^t and A_2^t, respectively over alphabets V1 and V2, let's give an intuitive definition. The product of A_1^t and A_2^t is a TA specifying a parallel system which may execute *all and only* the timed traces over the alphabet $V1 \cup V2$: (a) whose projections (Def.3.3) on V1 are accepted (Def.4.10) by A_1^t *and* ; (b) whose projections on V2 are accepted by A_2^t.

Definition 4.16. (Parallel product of two TA)

Let $A_i^t=(Qi,Vi,Ti,\mathcal{V}i,\delta i,qi_0)$, for i=1,2, be two TA over alphabets V1 and V2, with $T1 \cap T2=\emptyset$, and $\mathcal{V}i=\{Vci_1,...,Vci_{Nci}\}$. Their parallel product, noted $A_1^t \| A_2^t$, is defined by :

$$A_1^t \| A_2^t= (Q,V1 \cup V2,T1 \cup T2,\mathcal{V}1 \cup \mathcal{V}2,\delta,q_0) =Ext_{V1 \cup V2}(A_1^t) \times Ext_{V2 \cup V1}(A_2^t).$$ □

Remark 4.5. In Definition 4.16, if $V1 \subseteq V2$ then $A_1^t \| A_2^t=A_1^t \otimes A_2^t$, and if V1=V2 then $A_1^t \| A_2^t=A_1^t \times A_2^t$

Theorem 4.3. If $\mathcal{L}_{A_1^t}$ and $\mathcal{L}_{A_2^t}$ are the timed languages accepted by two TA A_1^t and A_2^t over alphabets V1 and V2, then : $\mathcal{L}_{A_1^t \| A_2^t}=\mathcal{L}_{Ext_{V1 \cup V2}(A_1^t)} \cap \mathcal{L}_{Ext_{V1 \cup V2}(A_2^t)}$ **(Proof :** *See Appendix A*). □

Property 4.4. If two TA A_1^t and A_2^t, respectively over alphabets V1 and V2, respect the finiteness property, then $A_1^t \| A_2^t$ respects the finiteness property. **(Proof :** *See Appendix A*). □

Example 4.3. Let $A_1^t=(Q1,V1,T1,\mathcal{V}1,\delta 1,q1_0)$ and $A_2^t=(Q2,V2,T2,\mathcal{V}2,\delta 2,q2_0)$ (Figure 7), with $\mathcal{V}i=\{Vci_1\}=\{Vi\}$, $Qi=\{qi_0,qi\}$, $Ti=\{ti_1,ti_2\}$, $Mc=Mci_1=10$, for i=1,2. $V1=\{a,b\}$, $V2=\{a,c\}$. Timers are $t1_1$, $t1_2$, $t2_1$ and $t2_2$, and counters are $c1_1$ and $c2_1$.

$\delta 1=\{[q1_0,a,q1,E1_1,\{t1_1\},K1],[q1,b,q1_0,E1_2,\{t1_2\},K1]\}$, $E1_1=(t1_1 \leq 5)$, $E1_2=(t1_1 \leq 2) \wedge (t1_2 \leq 5)$, $K1=(c1_1<10)$.

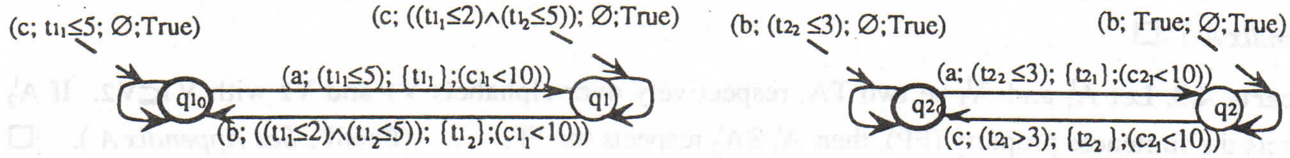
$\delta 2=\{[q2_0,a,q2,E2_1,\{t2_1\},K2],[q2,c,q2_0,E2_2,\{t2_2\},K2]\}$, $E2_1=(t2_2 \leq 3)$, $E2_2=(t2_1>3)$, and $K2=(c2_1<10)$.

$Ext_{V1 \cup V2}(A_1^t)$ and $Ext_{V2 \cup V1}(A_2^t)$ are on Figure 8, and the product of the two parallel TA is on Figure 9. The F_Conditions (Def. 4.4) of transitions in $A_1^t \| A_2^t$ (Fig. 9) are as follows.

Transitions with event a are enabled only if both $(c1_1<10)$ and $(c2_1<10)$ are true $(a \in Vc1_1 \cap Vc2_1)$.
Transitions with event b are enabled only if $(c1_1<10)$ is true (because $b \in Vc1_1$).
Transitions with event c are enabled only if $(c2_1<10)$ is true (because $c \in Vc2_1$).

7.a. A_1^t **7.b.** A_2^t

Figure 7. Two concurrent automata



8.a. $Ext_{V1 \cup V2}(A_1^t)$ **8.b.** $Ext_{V2 \cup V1}(A_2^t)$

Figure 8. Extensions of the two concurrent automata of Figure 7

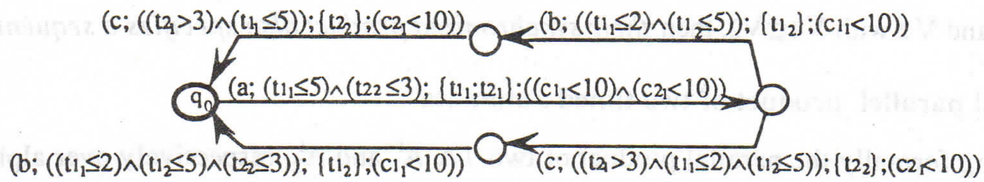


Figure 9. Synchronized product $A_1^t \| A_2^t$

5. Untimed automata

The problems we have encountered with timed automata, are the following :

(a) Respecting the timing requirements (T_Conditions and F_Conditions) does not ensure to avoid states respecting a given "indesirable" property, such as deadlock states;

(b) Finding and removing these "indesirable" states is not self-evident;

(c) Several processings (reductions, projections, minimization, ...) are not self-evident.

The approach we have used to tackle these problems consists in transforming a timed automaton into an equivalent untimed automaton (Def. 5.1) where transitions do not depend on parameters and where the event tick is represented by a transition. Therefore, all known methods used for FSMs can be used for untimed automata. Let's see two examples :
- We can remove deadlock states;
- An untimed automaton (UA) defined over an alphabet $W'=W \cup \{tick\}$ can be projected in any alphabet $V \subseteq W'$.

Thus, before making some processings, it may be convenient to transform a TA into a UA.

Definition 5.1. (Untimed automaton, operator *UntimeA*)
Let $A^t = A^t = (Q,V,T,\{V\},\delta,q_0)$ be a TA over an alphabet V which accepts (Def.4.10) a timed language $\mathcal{L}$, and let $\mathcal{L}^u = UntimeL(\mathcal{L})$. The untimed automaton $A^{ut}=(Q^{ut}, V \cup \{tick\}, \delta^{ut}, \langle q_0,0,0 \rangle)$ is the minimal FSM over the alphabet $V \cup \{tick\}$ which accepts the untimed language $\mathcal{L}^u$.

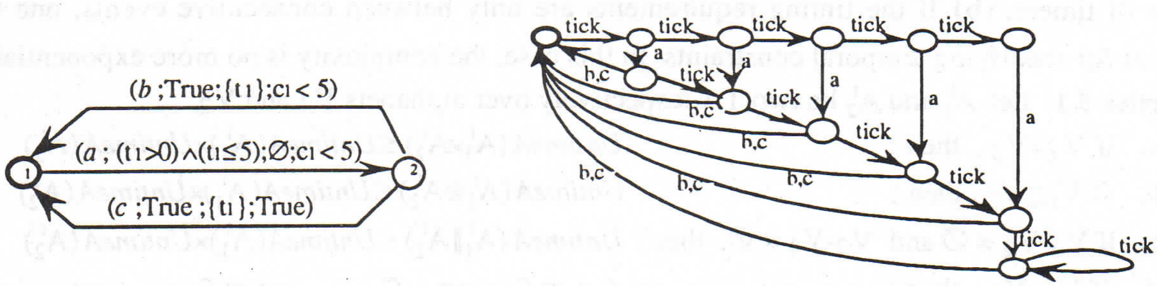
$\delta^{ut} \subseteq Q^{ut} \times V \cup \{ tick \} \times Q^{ut}$ defines the transitions of A^{ut}, and $\langle q_0,0,0 \rangle$ is the initial state of A^{ut}.
In other words : $\mathcal{L}_{A^{ut}} = UntimeL(\mathcal{L}_{A^t})$, (Def.3.6, for *UntimeL*), and A^t and A^{ut} are called equivalent.
A sufficient condition of existence of A^{ut} is the finiteness of the sets of timers and counters.
We also define the surjective operator *UntimeA* such that : $A^{ut} = UntimeA(A^t)$. □

Example 5.1. Let's consider the timed $A^t = (Q,V,T,\{V\},\delta,q_0)$ on Figure 10.a, where we use one timer t_1, and one counter c_1 w.r.t. V. Since $Mt_1=5$ and $Mc_1=5$, t_1 is smaller than or equal to 6, and c_1 is smaller than or equal to 5. The obtained untimed A^{ut} is on Figure 10.b, each state being defined by $\langle q,t_1,c_1 \rangle$, where q is a state in A^t.

Remark 5.1. Since untimed traces accepted by A^{ut} correspond to infinite timed traces accepted by A^t, then A^{ut} accepts only infinite untimed traces, and does not contain indesirable states. An indesirable state is either a deadlock state or a state from which only a selfloop *tick* is executable.

- A deadlock in A^{ut} is indesirable, because it has no sense. In fact, a deadlock state means that the event *tick* is not executable. Therefore, the passing of time is stopped!
- A state from which only a selfloop *tick* is executable is indesirable, because it implies that A^{ut} accepts a trace TRC=*UntimeT*(Trc) where Trc is a **finite** timed trace!

Informally, A^{ut} allows to represent a real-time system specified by A^t, as a system without timing requirement, but where a new event *tick* is added. This event, which models the passing of one unit of clock time (uct), is processed like any other event.



10.a. A^t 10.b. A^{ut}
Figure 10. Timed and untimed automata

Let's give an idea of how A^{ut} is obtained from A^t over an alphabet V, when only one counter c_1, w.r.t. $Vc_1=V$ is used. This implies that A^t respects the finiteness property (Property 4.1). Let $T=\{t_1,..., t_{Nt}\}$ be a set of timers used for defining A^t, let Mt_i be the maximum value a timer t_i is compared to, for defining the T_Conditions (Def. 4.2) of all the transitions of A^t. In this case, t_i does not need to be incremented as soon as $t_i=Mt_i+1$ (Def. 4.7). A state of A^{ut} is defined by $\langle q1,ts,c_1 \rangle$, where q1 is a state of A^t, $ts=(t_1,...,t_{Nt})$ is a timer state (Def. 4.1). The passing of one uct is represented in A^{ut} by the event *tick*. Execution of *tick* from state $\langle q1,ts,c_1 \rangle$ leads to state $\langle q1,ts+1,0 \rangle$, i.e., timers are incremented and the counter is set to zero. Execution of an event $\sigma \neq tick$ from state $\langle q1,ts,c_1 \rangle$ of A^{ut} leads to state $\langle q2,ts',c_1+1 \rangle$, where q2 is a state of A^t which is reached by a transition tr=[q1;σ;q2;E;R;K] from state q1 of A^t (with E and $K=(c_1<Mc_1)$ are equal to TRUE for the current timer state ts, and $c_1<Mc_1$), and ts' is

obtained from ts by setting to zero timers belonging to R. Besides, A^{ut} is minimal and does not contain indesirable states.

Remark 5.2. (a) if ts=($Mt_1+1, \ldots, Mt_{Nt}+1$) then ts+1=ts. In this case, an event *tick* is a selfloop in A^{ut}; (b) Since two A_i^{ut} over alphabets $V_i \cup \{tick\}$, for i=1,2, are FSM, we can use the classic synchronized product between them, noted $A_1^{ut} \times A_2^{ut}$, where events of $(V_1 \cap V_2) \cup \{tick\}$ are executed conjointly.
(c) The product $UntimeA(A_1^t) \times UntimeA(A_2^t)$ may contain deadlocks, therefore it does not correspond to a real DES. In fact, a deadlock prevents the event *tick*, i.e., the passing of time is stopped.

Lemmas 5.1. Let $A^t = A_i^t = (Q, V, T, \mathcal{V}, \delta, q_0)$ be a TA and $A^{ut} = UntimeA(A^t) = (Q^{ut}, V \cup \{tick\}, \delta^{ut}, \langle q_0, 0, 0 \rangle)$. Let's remind some notations : (a) Nt and Nc are the numbers of timers and counters; (b) Mc bounds all the Mc_i, for i=1,...,Nc; (c) Mt is the maximum constant any timer is compared to; (d) $|Q|$ and $|\delta|$ are numbers of states and of transitions of A^t.
5.1.a. The number $|Q^{ut}|$ of states of A^{ut} is bounded by : $|Q| * (Mt+2)^{Nt} * (Mc+1)^{Nc}$
5.1.b. The number $|\delta^{ut}|$ of transitions of A^{ut} is bounded by : $(|Q| + |\delta|) * (Mt+2)^{Nt} * (Mc+1)^{Nc}$
5.1.c. The complexity for calculating A^{ut} is in : $O(|Q^{ut}|^2) = O(|Q|^2 * (Mt+2)^{2 \times Nt} * (Mc+1)^{2 \times Nc})$.
$|Q^{ut}|$, $|\delta^{ut}|$ and the complexity for calculating A^{ut} are then exponential in the numbers of timers and of counters. (Proof : *See Appendix A*). □

Remark 5.3. (a) The number of counters is not really a problem. In fact, in general one counter is sufficient, for ensuring the finiteness property. Therefore, the complexity is essentially due to the number of timers. (b) If the timing requirements are only between consecutive events, one timer is sufficient for specifying temporal constraints. In this case, the complexity is no more exponential.
Properties 5.1. Let A_1^t and A_2^t be two TA respectively over alphabets V_1 and V_2.

5.1.a. If $V_1 = V_2$, then : $UntimeA(A_1^t \times A_2^t) \leq UntimeA(A_1^t) \times UntimeA(A_2^t)$

5.1.b. If $V_1 \subseteq V_2$, then : $UntimeA(A_1^t \otimes A_2^t) \leq UntimeA(A_1^t) \times UntimeA(A_2^t)$

5.1.c. If $V_1 - V_2 \neq \emptyset$ and $V_2 - V_1 \neq \emptyset$, then : $UntimeA(A_1^t \| A_2^t) \leq UntimeA(A_1^t) \times UntimeA(A_2^t)$

5.1.d. If $V_1 = V_2$, then : $\mathcal{L}_{A_1^t} \subseteq \mathcal{L}_{A_2^t} \Rightarrow \mathcal{L}_{UntimeA(A_1^t)} \subseteq \mathcal{L}_{UntimeA(A_2^t)}$

5.1.e. If $V_1 \subseteq V_2$, then : $UntimeL(Proj_{V_1}(\mathcal{L}_{A_2^t})) = Proj_{V_1}(UntimeL(\mathcal{L}_{A_2^t}))$

(where $A \leq B$ means $\mathcal{L}_A \subseteq \mathcal{L}_B$) **Proof** : *See Appendix A* □

6. Temporized automata to specify a protocol and timing requirements on the medium

As it is mentioned at the beginning of Section 5, the use of untimed automata is convenient to make several processings. This fact is taken into account by the procedures of protocol synthesis in Sections 7 and 8, where timed automata are transformed into untimed automata (Sect. 5), before making some computations on the specifications. The results of these procedures are then untimed automata which specify modules to be implemented.

The problem with untimed automata is that they are not convenient to specify systems to be implemented, because the number of states and transitions may be very important (Lemmas 5.1.a and 5.1.b). Our aim is then to transform every untimed automaton A^{ut} into an equivalent and more concise automaton. The first idea which comes into the mind is to compute a timed automaton A^t such that $A^{ut}=UntimeA(A^t)$ (Def. 5.1). The main problems with timed automata are:

- the respect of timing requirements in a TA may lead to deadlocks;
- the computation of a TA A^t such that $A^{ut}=UntimeA(A^t)$, is not self-evident.

Therefore, we propose a second model based on *temporized automata* where the above problems do not exist. Temporized automata are not used to specify a desired service and a supremal behaviour of the medium (Sect. 4), because they are less intuitive than timed automata, and then cannot be easily computed from an unformal specification.

6.1. Temporized automata

Before defining formally a temporized automaton, let's give an intuitive idea. A temporized automaton A^{tp} uses a variable ι and a timer t. The enabling condition of every transition of A^{tp} depends on ι and t. When a transition occurs, the current values of t and ι may change.

Definition 6.1. (Timer t, variable ι)
Timer t is a variable which belongs to a finite set $\mathcal{T}=\{0, 1, ..., t_{max}\}$ of natural numbers.
t is automatically incremented by one with every tick, if its value is smaller than t_{max}.
The variable ι is a variable which belongs to a finite set $\mathcal{I}=\{1, 2, ..., i_{max}\}$ of natural numbers..
t and ι can be set with the occurrence of any transition. □

Let $A=(Q,V,\delta,q_0)$ be a FSM where Q is a set of states, V is an alphabet, q_0 is the initial state, and $\delta \subseteq Q \times V \times Q$ defines the transitions, i.e., a transition of A can be represented by $[q,\sigma,r]$. Let's see how a temporized automaton can defined from the FSM A.

Definition 6.2. (Transformation function, temporized transition)
Let $\mathcal{T}$ and $\mathcal{I}$ be respectively the sets of values of timer t and variable ι.
A *transformation function*, w.r.t. $\mathcal{T}$ and $\mathcal{I}$, is any function : $\mathcal{I} \times \mathcal{T} \rightarrow \mathcal{I} \times \mathcal{T}$. Let then $\mathcal{A}$ be the set of transformation functions, w.r.t. $\mathcal{T}$ and $\mathcal{I}$.
A *temporized transition*, w.r.t. A and $\mathcal{T}$ and $\mathcal{I}$, is defined by $Tr=[q_1,\sigma,q2;A]$, where $[q_1,\sigma,q2] \in \delta$ and $A \in \mathcal{A}$. The semantics of Tr is the following.
Let q_1 be the current state, i_1 be current value of ι, and t_1 be the current value of t.
(1) σ may occur only if $A(i_1,t_1)$ is defined;
(2) σ *must* occur if $A(i_1,t_1)$ is defined and if $A(i_1,t)$ is not defined for any $t>t_1$.
(3) after the occurrence of σ : (a) the state q_2 is reached;
(b) timer t is set to t_2, and variable ι is set to i_2, where $A(i_1,t_1)=(i_2,t_2)$. □

A temporized automaton A^{tp} can then be constructed if we transform every transition $tr=[q_1,\sigma,q_2]$ of A into a temporized transition Tr by associating to it a transformation function.

Definition 6.3. (Temporized automaton)
Formally, a temporized automaton $A^{tp}=(Q, V, \delta^{tp}, \mathcal{I}, \mathcal{T}, q_0, i_0, t_0)$ is defined as follows. Q is a set of states, V is the alphabet, $\mathcal{I}$ and $\mathcal{T}$ are respectively the sets of values of ι and t, q_0 is the initial state, i_0

is the initial value of ι, t_0 is the initial value of $\mathfrak{t}$, and $\delta^{tp} \subseteq Q \times V \times Q \times \mathcal{A}$ defines the temporized transitions (Def. 6.2). □

Definition 6.4. (Acceptance of a timed trace and of a language, equivalence, partial order relation)
Let $A^{tp}=(Q, V, \delta^{tp}, \mathcal{I}, \mathcal{T}, q_0, i_0, t_0)$ and let $Trc=\langle\sigma_1,\tau_1\rangle...\langle\sigma_i,\tau_i\rangle...$ be an infinite timed trace.
Let $\mathcal{Tr}=Tr_1 Tr_2 ... Tr_i ...$ be an infinite sequence of transitions of A^{tp}, with :

$$\forall i \in \mathbb{N}^* : Tr_i=[q_{i-1},\sigma_i,q_i;A_i] \in \delta^{tp}.$$

- The infinite timed trace $Trc=\langle\sigma_1,\tau_1\rangle...\langle\sigma_i,\tau_i\rangle...$ *is accepted by* $\mathcal{Tr}$, if and only if :

  For all $i>0$: $u_i=c_{i-1}$, $v_i=t_{i-1}+\tau_i-\tau_{i-1}$, and $(c_i,t_i)=A_i(u_i,v_i)$; where $\tau_0=0$, $c_0=i_0$.
- The infinite timed trace $Trc=\langle\sigma_1,\tau_1\rangle...\langle\sigma_i,\tau_i\rangle...$ *is accepted by* A^{tp} if and only if there exists an infinite sequence $\mathcal{Tr}$ of transitions of A^{tp} which accepts Trc.
- A timed language, noted $\mathcal{L}_{A^{tp}}$, *is accepted by* A^{tp} if it contains all and only the traces accepted by A^{tp}. Informally, a system modeled by A^{tp} may execute a trace accepted by A^{tp}.
- A_1^{tp} and A_2^{tp} are *equivalent*, and noted $A_1^{tp} \equiv A_2^{tp}$, if and only if $\mathcal{L}_{A_1^{tp}} = \mathcal{L}_{A_2^{tp}}$.
- A_1^{tp} *is smaller than or equal to* A_2^{tp}, and noted $A_1^{tp} \leq A_2^{tp}$, if and only if $\mathcal{L}_{A_1^{tp}} \subseteq \mathcal{L}_{A_2^{tp}}$. □

Definition 6.5. (Operator *Temp*)
Let A^{ut} be an untimed automaton accepting the untimed language $\mathcal{L}_{A^{ut}}$, we define the operator *Temp* by:

$$A^{tp}=Temp(A^{ut}) \Leftrightarrow \mathcal{L}_{A^{tp}}=TimeL(\mathcal{L}_{A^{ut}}) \Leftrightarrow \mathcal{L}_{A^{ut}}=UntimeL(\mathcal{L}_{A^{tp}}) \quad \text{(See Def. 3.6).}$$

If $A^{tp}=Temp(A^{ut})$ then A^{tp} and A^{ut} are called equivalent. □

6.2. Transformation from an untimed automaton to a temporized automaton

Since a temporized automaton A^{tp} is automatically computed from an untimed automaton A^{ut}, let's show in a simple example the principle of the transformation from A^{ut} to A^{tp}. The untimed automaton considered is represented on Figure 11.a, and our aim is to transform it into an equivalent temporized automaton modeling the same behaviour (Def. 6.5). The main steps to transform A^{ut} into A^{tp} are semiformally enumerated below.

Step 1: *Defining states of* A^{tp}

To each subset $\mathcal{S}$ of states of A^{ut} which are closed under tick, corresponds one state S in A^{tp}. Let then:

- Q^{ut} and Q be respectively the sets of states of A^{ut} and A^{tp},
- $2^{Q^{ut}(tick)}$ be the set of subsets of Q^{ut} closed under tick,
- The bijection $P_{A^{ut}} : 2^{Q^{ut}(tick)} \rightarrow Q$, such that $P_{A^{ut}}(\mathcal{S})=S$.

In the example (Fig. 11.a), $Q^{ut}=\{1, 2, ..., 13\}$, $2^{Q^{ut}(tick)}=\{\mathcal{A}, \mathcal{B}, \mathcal{C}\}$, where $\mathcal{A}=\{1,2,3\}$, $\mathcal{B}=\{4,5\}$, $\mathcal{C}=\{6,..., 13\}$, and then $Q=\{A, B, C\}$, with $P_{A^{ut}}(\mathcal{A})=A$, $P_{A^{ut}}(\mathcal{B})=B$, $P_{A^{ut}}(\mathcal{C})=C$.

Step 2: *Relabeling the states of* A^{ut}

All the states of A^{ut} are renamed as follows.

For every subset $\mathcal{S}$ of states closed under tick, i.e., associated to a same state S of A^{tp} :

- Let N_S be the number of states $s_1, ..., s_{N_S}$ of $\mathcal{S}$ without an ingoing tick ;

- Every state of $\mathcal{S}$ is associated to an N_S-uplet. The latter is computed with the three following rules;

  R1 : A state s_i (without an ingoing tick) is associated to the N_S-uplet $(t_1, ..., t_j, ..., t_{N_S})$ with $t_i = 0$, and $t_j = \lambda$ if $j \neq i$, where λ is a negative value.

  R2 : Let e be a state of $\mathcal{S}$ associated to an N_S-uplet $(t_1, ..., t_{N_S})$. For every state s_i (with $i \leq N_S$), if e cannot be reached from s_i by a sequence of ticks then : $t_i = \lambda$.

  R3 : Let e and f be two different states of $\mathcal{S}$, respectively associated to the N_S-uplets $(t_1,...,t_{N_S})$ and $(u_1,...,u_{N_S})$. If f is reached from e after a tick then : $u_i=t_i+1$ if $t_i \neq \lambda$, for $i=1, ..., N_S$.

- Every state e of $\mathcal{S}$ is then relabeled by (S;t), where S is the same for all states of $\mathcal{S}$, and t is an N_S-uplet computed with the the three above rules.
- If $t = (t_1,...,t_{N_S})$, then every t_i is called the component i of t, if $t_i \neq \lambda$.
- Let then $\mathcal{T}_S$ be the set of all N_S-uplets associated to the states of S; $\mathcal{T}_{S_i}$ (with $i \leq N_S$) is then the set of components i of elements of $\mathcal{T}_S$.

The relabeling of states of A^{ut} of Figure 11.a is represented on Figure 11.b, with :
- $N_A=1$, $N_B=1$, $N_C=2$,
- $\mathcal{T}_A=\{0, 1, 2\}$, $\mathcal{T}_B=\{0, 1\}$, $\mathcal{T}_C=\{(0, \lambda), (1, \lambda), (2, \lambda), (3, \lambda), (\lambda, 0), (\lambda, 1), (\lambda, 2), (4, 3)\}$,
  $\mathcal{T}_{C_1}=\{0, 1, 2, 3, 4\}$, $\mathcal{T}_{C_2}=\{0, 1, 2, 3\}$.

Step 3 Computing transitions of A^{tp}

For every pair $\mathcal{S}_i$ and $\mathcal{S}_f$ of sets of states closed under tick :
- Let S_i and S_f be the corresponding states in A^{tp} (see Step 1);
- All transitions executing a same event σ, from any state of $\mathcal{S}_i$ towards any state of $\mathcal{S}_f$, are represented by a same transition $Tr=[S_i,\sigma,S_f;A]$ in A^{tp}, where A is a transformation function (Def. 6.2) computed as follows.
- If a transition $tr=[(S_b,t),\sigma,(S_f,u)]$ is defined in A^{ut}, with $t=(t_1, ..., t_{N_{S_b}})$ and $u=(u_1, ..., u_{N_{S_f}})$ then : for any $i \leq N_{S_b}$ such that $t_i \neq \lambda$, $A(i,t_i)=(k,u_k)$, where k is the smallest indice such that $u_k \neq \lambda$.

For the example of Figure 11.b, we obtain the temporized automaton $A^{tp}=(Q, V, \delta^{tp}, \mathcal{T}, \mathcal{I}, q_0, t_0, i_0)$ represented on Figure 12, with :

$\mathcal{I}=\{1,2\}$ because two sequences of ticks (from states 6 and 10 of Fig. 11.a) are closed under tick.

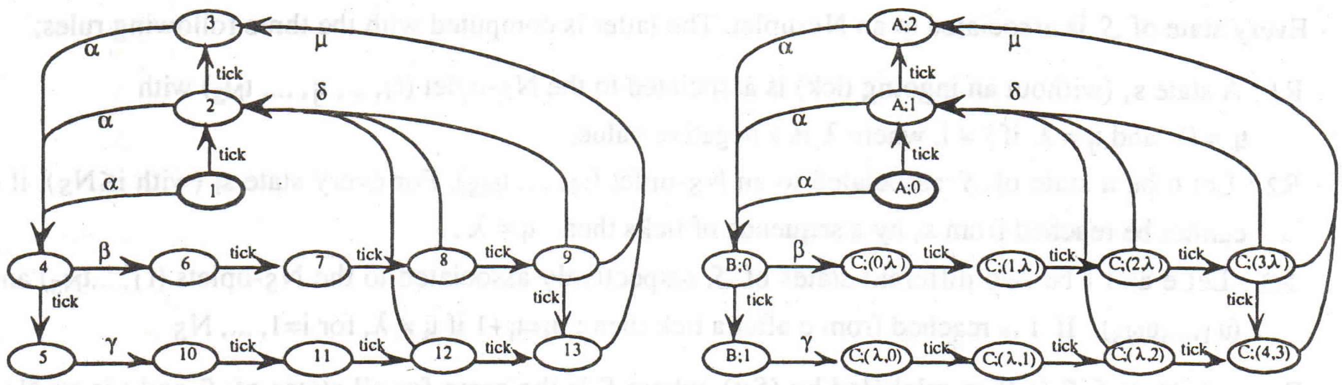
$\mathcal{T}=\{0,1, 2, 3, 4, 5\}$ because 4 the biggest length of a sequence of ticks.

$q_0=A$, $i_0=1$, $t_0=0$, and the transformation functions are :

$A_1(1,0)=A_1(1,1)=A_1(1,2)=(1,0)$, $A_2(1,0)=(1,0)$, $A_3(1,1)=(2,0)$,

$A_4(1,2)=A_4(1,3)=A_4(2,2)=(1,1)$, and $A_5(1,3)=A_5(1,4)=A_5(2,3)=(1,2)$.

Let's mention that a temporized transition $Tr=[q_1,\sigma,q_2;A]$ is represented graphically by : $q_1 \xrightarrow{(\sigma;A)} q_2$



11.a. Untimed automaton A^{ut} 11.b. Relabeling states of A^{ut} with values of timers

Figure 11. Renaming states of A^{ut} before transforming it into a temporized automaton

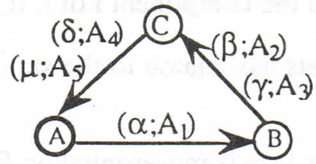


Figure 12. Temporized automaton corresponding to the untimed Aut of Figure 11

Property 6.1. Let A^{ut} and B^{ut} be two untimed automata over a same alphabet V.

$$A^{ut} \leq B^{ut} \Leftrightarrow Temp(A^{ut}) \leq Temp(B^{ut})$$

(where $Temp(A^{ut}) \leq Temp(B^{ut})$ means $\mathcal{L}_{Temp(A^{ut})} \subseteq \mathcal{L}_{Temp(B^{ut})}$). **Proof** : *See Appendix A* □

Lemma 6.1. Let A^{ut} be an untimed automaton and $A^{tp} = Temp(A^{ut})$. Let $|Q^{ut}|$ and $|\delta^{ut}|$ be respectively numbers of states and of transitions of A^{ut}. The complexity for calculating A^{tp} from A^{ut} is in :

$O(|Q^{ut}|*|V|*\log_2(|Q^{ut}|*|V|) + |\delta^{ut}|*|Q^{ut}|^2*\log_2(|Q^{ut}|))$ (**Proof** : *See Appendix A*). □

7. Protocol derivation for sequential real-time systems

The two starting points of the protocol derivation are (Sect. 2) : (a) a specification of the desired service; (b) a model of the supremal behaviour of the medium. They are specified with timed automata (Sect. 4). The results of the protocol derivation are the specifications of (Sect. 2): (i) the protocol in each site of the distributed system; (j) the timing requirements on the medium. They are specified with temporized automata (Sect. 6).

In the present section, we propose a procedure of protocol derivation when the desired service is sequential and specified by one timed automaton. Before presenting the main steps of the procedure, let's give the following definition.

Definition 7.1. (outgoing, ingoing, out(q), in(q), $outst_i(q)$, nbrout(q))
Let SS^t be a TA specifying a desired service, and let q be one of its states.

Outgoing (resp. *ingoing*) transitions of q are transitions which are executable from (resp. lead to) q.

out(q) (resp. in(q)) contains identifiers of sites where outgoing (resp. ingoing) transitions of q occur.

$outst_i(q)$ is the set of states of SS^t reachable from q by transitions executed by PE_i.

nbrout(q) is the number of transitions executable from q. □

Example 7.1. For SS^t of Figure 3.b (Sect. 4.1), in(1)={2}, in(2)={1}, out(1)={1},out(2)={2}, $oust_1(1)=\{2\}$, $oust_2(1)=\emptyset$, $oust_1(2)=\emptyset$, $oust_2(2)=\{1\}$, nbrout(1)=nbrout(2)=1. □

As mentioned in Section 2, the principle of the protocol synthesis is the following.

If after execution of a primitive A_a by a protocol entity PE_a, there is a choice between several primitives executed by different PE_{bi}, for i=1,2,..., p, then :
When PE_a executes the primitive A_a, it selects one PE_{bi} and sends a message to it to inform it that it may execute one of its primitives. This principle implies the two following rules.

Rule 1. The outgoing transitions (Def. 7.1) of the initial state q_0 of SS^t are executable by *a same protocol entity*, i.e., cardinal of out(q_0) is equal to one ($|out(q_0)|=1$). □

Informally, Rule 1 requires that the first action of the desired service is always executed by a same site.

Rule 2. After execution of a primitive A_a by PE_a, the choice between several primitives executed by different PE_{bi}, for i=1,2,..., p, is achieved in two steps.

- First Step : PE_a selects one PE_{bi} ;
- Second Step : The PE_{bi} selected chooses one its primitives. □

7.1. Transformation of the service specification

The first thing to do is to transform SS^t into another timed automaton TSS^t (Transformed SS^t) with the following rules.

First step : each timed transition of SS^t: $q1 \xrightarrow{(A_k;\, E(ts);\, R;K(cs))} q2$

is replaced by : $q1 \xrightarrow{(A_k;\, E(ts);\, R;K(cs))} r \xrightarrow{i(q2)} q2$

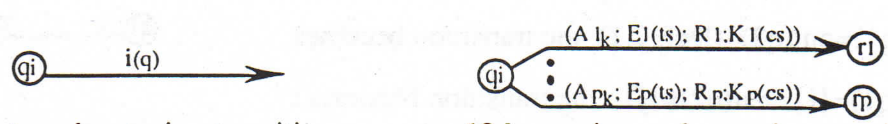
A new state r is then inserted between each pair of states q1 and q2 connected by a transition. r and q2 are connected by an internal transition i(q2) parameterized by q2.

After this first step, we obtain a TA noted TS^t. Let's notice that if a state of TS^t is reachable by an internal transition i(q), then its outgoing transitions are not internal.

Second step : The specification TS^t is transformed into an equivalent TSS^t, such that every state qi of TSS^t respects either condition C1 or condition C2, defined below.

C1 = only an internal transition i(q) is executable from qi (Fig. 13.a),

C2 = no internal transition is executable from qi, and all outgoing transitions (Def.7.1) of qi are executable by a same protocol entity, i.e., cardinal of out(qi) is equal to one (|out(qi)|=1, Fig.13.b). On Figure 13.b., out(qi)={k} and $outst_k(qi)=\{r1, \ldots, rp\}$.



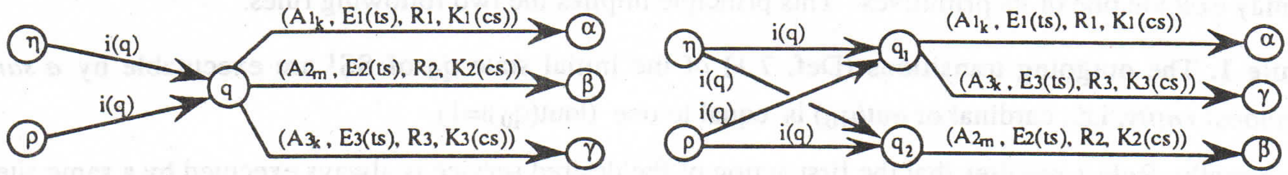
13.a. internal outgoing transition **13.b.** non internal outgoing transitions

Figure 13. Outgoing transitions in a state of the transformed specification TSS^t.

The way for obtaining TSS^t from TS^t is the following. Every state q of TS^t reachable by internal transition(s) (Fig.14.a), is replaced by as many states qi as the cardinal of out(q) (Fig.14.b) . Outgoing transitions of states qi (which are not internal) must respect the preceding condition C2, and the following condition C3. Ingoing transitions of states qi must respect the following condition C4.

C3 : Outgoing transitions of two different states qi and qj of TSS^t (Fig.14.b), generated from a same state q of TS^t (Fig.14.a), are executed by two different protocol entities.

C4 : The sets of ingoing transitions (which are internal) of two different states qi and qj of TSS^t, generated from a same state q of TS^t, are equal to the set of ingoing transitions of state q (Fig.14).



14.a. State e in TS^t **14.b.** Transformation of e in TSS^t

Figure 14. Example of transformation from TS^t to TSS^t

Remark 7.1.(a) if two states r1 and r2 of TSS^t are connected by a transition i(q) then $|in(r1)|=|out(r2)|=1$; **(b)** if $TSS^t \neq TS^t$, then TSS^t is non deterministic; **(c)** if for every state q of SS^t, $|ou(q)|=1$, then $TSS^t=TS^t$.

Definition 7.2. (Operator *Transf*)

Operator *Transf* is simply defined by : TSS^t=*Transf* (SS^t). □

Example 7.2. SS^t of Figure 3.b (Sect. 4.1) is transformed into TSS^t of Figure 15. In this example, only the first step of the transformation is used, because $|ou(1)|=|ou(2)|=1$ (Remark 7.1.c).

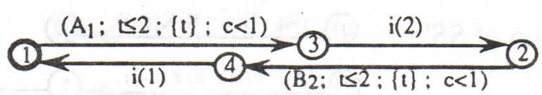


Figure 15. Transformation of SS^t of Figure 3.b.

7.2. Procedure of protocole derivation for a sequential desired service

The entries of the procedure are : (a) a TA SS^t specifying a sequential desired service; (b) For each pair (PE_i,PE_j), a TA $SupMed^t_{i,j}$ (Sect. 4.2) specifying the supremal behaviour of the medium.

The proposed procedure of protocol derivation, is called *Der_Seq_Prot* and consists of ten steps.

Step 1 : SS^t is transformed into TSS^t, i.e., TSS^t=*Transf* (SS^t) (Sect. 7.1, Def. 7.2).

Step 2 : From TSS^t and the different $SupMed^t_{i,j}$, we generate $MedSS^t_\varepsilon$ with the following rules :

- A not internal transition remains unchanged.
- An internal transition i(q) $q1 \xrightarrow{i(q)} q2$ is replaced by :

Case a : if in(q1)=out(q2) (Def. 7.1), the transition becomes : $q1 \xrightarrow{\varepsilon} q2$

Case b : if in(q1)={i}≠out(q2)={j}, the transition becomes :

$$q1 \xrightarrow{(s^j_i(q);True;\ \{t_{i,j}\};\ True)} \bigcirc \xrightarrow{(r^i_j(q);\ E_{i,j}(t_{i,j});\ \varnothing;\ True)} q2$$

The transformation of Step 2 uses $SupMed^t_{i,j}$ (Sect. 4.2), but with s^j_i and r^i_j parameterized by q.

Informally, i(q) consists in : (a) doing nothing, if it connects two consecutive transitions of SS^t executed by a same PE_i ; (b) sending a message from PE_i to PE_j, if it connects two consecutive transitions of SS^t respectively executed by PE_i and PE_j. The message is parameterized by q.

Step 3 : Transitions ε of $MedSS^t_\varepsilon$ are removed by projection for obtaining $MedSS^t$. An algorithm for removing these ε is proposed in [2].

Step 4 : $MedSS^t$ is untimed (Def. 5.1) for obtaining $MedSS^{ut}$=*UntimeA*($MedSS^t$). $MedSS^{ut}$ is a minimal FSM containing the event *tick*. Let's notice that the four following steps process FSMs with event *tick* .

Step 5 : We generate an untimed automaton GPS^{ut} (global protocol specification), by adding a second parameter to each event $s_i^j(q)$ or $r_j^i(q)$ in $MedSS^{ut}$, with the following rule :

A transition $q1 \xrightarrow{s_i^j(q)} q2$ is replaced by a transition $q1 \xrightarrow{s_i^j(q,q2)} q2$. The same transformation is made on transitions $r_j^i(q)$. This transformation allows to differentiate two transitions $s_i^j(q)$ (or $r_j^i(q)$) which do not lead to the same state in $MedSS^{ut}$.

Intuitively, if a primitive A_i is executed by a protocol entity PE_i, and is followed by execution of a primitive B_j by PE_j, then after execution of A_i by PE_i, this one sends a message to PE_j to inform it that it may execute B_j. When there is no timing requirement, the message contains only one parameter q which informs PE_j about the primitive which has been executed. When there are timing requirements, PE_i sends a message with a second parameter q2 (event $s_i^j(q,q2)$); the latter informs the medium about the delay t_1 between A_i and $s_i^j(q,q2)$. When the message reaches its destination, the medium replaces q2 by r2 (event $r_j^i(q,r2)$); the latter informs PE_j about $t=t_1+t_2$, where t_2 is the transit delay of the message in the medium.

Step 6 : For each PE_i, the untimed automaton PS_i^{ut} is derived by projecting GPS^{ut} in the alphabet $V_i \cup \{$ *tick* $\}$, where V_i contains all events in GPS^{ut} executed by PE_i. An event of V_i may correspond to : (a) execution of a primitive by PE_i; (b) an event $s_i^j(q,q2)$; (c) an event $r_i^k(q,r2)$, with $j,k \neq i$.

Step 7 : For each pair (PE_i, PE_j) and each q, where PE_i sends to PE_j a message whose first parameter is q (i.e., events $s_i^j(q,*)$ and $r_j^i(q,*)$ exist in GPS^{ut}), the untimed automaton $ReqMed_{i,j}^{ut}(q)$ is generated by projecting GPS^{ut} in the alphabet $V_{i,j}(q) \cup \{$ *tick* $\}$. An element of $V_{i,j}(q)$ may be any event $s_i^j(q,*)$ and $r_j^i(q,*)$ of GPS^{ut}. The obtained $ReqMed_{i,j}^{ut}(q)$ specifies the behaviour of the medium when it carries, from PE_i to PE_j, a message whose first parameter is q.

The informal semantics of the different PS_i^{ut} (Step 6) and $ReqMed_{i,j}^{ut}(q)$ (Step 7) is the following. If the different protocol entities PE_i are specified by PS_i^{ut}, and if the medium respects the specifications $ReqMed_{i,j}^{ut}(q)$, then the service SS^t is totally or partially provided (Def.7.3 and 7.4).

Step 8 : The systems specified by PS_i^{ut} and $ReqMed_{i,j}^{ut}(q)$ - obtained at steps 6 and 7 - must be in their initial states simultaneously, when the discrete time τ (Sect. 3.1) is initialized to zero. In another words, the specifications PS_i^{ut} and $ReqMed_{i,j}^{ut}(q)$ are relative to an absolute time. This implies that the local clocks in all sites of the distributed system are synchronized and then equivalent to a global clock. The aim of the present step is to transform the specifications in such a way that the local clocks do not need to be synchronized.

Let's remark that there is a redundancy in timings constraints of the specifications PS_i^{ut} and $ReqMed_{i,j}^{ut}(q)$. In fact : (a) Since timing constraints on sendings of messages are specified on PS_i^{ut}, they do not need to be specified on $ReqMed_{i,j}^{ut}(q)$; (b) Since timing constraints on receptions of messages are specified on $ReqMed_{i,j}^{ut}(q)$, they do not need to be specified on PS_i^{ut}. Therefore, the transformation which consists in removing timing constraints on : - Receptions of messages from any PS_i^{ut};

- Sendings of messages from any $ReqMed_{i,j}^{ut}(q)$.

does not modify the service provided to the user (Def. 7.3).

Besides, with this transformation the local clocks of the different sites do not need to be synchronized.

More formally, the transformation consists in :

For every PS_i^{ut} and any $k \neq i$:

- a sequence of ticks which precedes a transition $r_i^k(q,r)$ is replaced by a sequence of ε;
- a selfloop tick is added to any state from which a transition $r_i^k(q,r)$ executable.

For every $ReqMed_{i,j}^{ut}(q)$:

- a sequence of ticks which precedes a transition $s_i^j(q,r)$ is replaced by a sequence of ε;
- a selfloop tick is added to any state from which a transition $s_i^j(q,r)$ executable.

After this transformation, transitions ε are removed by projection.

Step 9 : The untimed automata PS_i^{ut} and $ReqMed_{i,j}^{ut}(q)$ obtained at Step 8 are transformed into temporized automata, by using operator *Temp* (Def. 6.5). Therefore :

- Every protocol entity PE_i is then specified by $PS_i^{tp}=Temp(PS_i^{ut})$;
- The behaviour of the medium, when it carries from PE_i to PE_j a message whose first parameter is q, is specified by $ReqMed_{i,j}^{tp}(q)=Temp(ReqMed_{i,j}^{ut}(q))$.

Step 10 : The temporized automata are transformed as follows.

For any i,j,q : all transitions $[q_1, s_i^j(q,*), q_2, ?]$ (where $*$ is any parameter and ? is any function) are represented by one transition $[q_1, s_i^j(q,x), q_2, f_x]$, where x is a variable and f_x is a transformation function depending on the value of x. The same transformation is made on transitions $[q_1, r_i^j(q,*), q_2, ?]$.

An example of this transformation is given in Section 7.3.

End of *Der_Seq_Prot* ■

Let's remark that Steps 8, 9 and 10 are closely related to the three important contributions (mentioned in Abstract and Section 1.2) of this paper. In fact :

- In Step 8, the untimed specifications obtained by the protocol synthesis are optimized in the sense that they do not necessitate to synchronize the different local clocks of each site of the distributed system (Sect. 7 and 8).
- Step 9 uses temporized automata which are formally defined in Section 6;

- In Step 10, the temporized specifications obtained by the protocol synthesis are improved in the sense that they are more concise : several transitions are represented by one parameterized transition (Example in Section 7.3).

Definition 7.3. (Provided service $PrSS^{ut}$)
For computing an untimed automaton (with event *tick*), noted $PrSS^{ut}$, which models the service provided to the user, one only has to project $MedSS^{ut}$ (Step 4) in $V \cup \{tick\}$, where V is the alphabet of SS^t. Informally, this projection consists in keeping visible, in sequences accepted by $MedSS^{ut}$, only events of SS^{ut}. □

Theorem 7.1. If SS^t specifies a desired service, let $SS^{ut}=UntimeA(SS^t)$ (Def.5.1), and let $PrSS^{ut}$ be the specification of the provided service. Then : $PrSS^{ut} \leq SS^{ut}$ (i.e., $\mathcal{L}_{PrSSut} \subseteq \mathcal{L}_{SSut}$).
The safety is then ensured. (**Proof :** *See Appendix A* . □

Definition 7.4. (Service totally or partially provided)
Let SS^{ut} and $PrSS^{ut}$ be untimed automata specifying respectively the desired and the provided service.
The service is said *totally provided* if and only if : $SS^{ut} \equiv PrSS^{ut}$, i.e., $\mathcal{L}_{PrSSut} = \mathcal{L}_{SSut}$,
The service is said *partially provided* if and only if : $SS^{ut} < PrSS^{ut}$, i.e., $\mathcal{L}_{PrSSut} \subset \mathcal{L}_{SSut}$. □

7.3. Example

We consider the desired specified by SS^t of Figure 3.b (Sect. 4.1), and the supremal behaviour of the medium modeled by $SupMed^t_{1,2}$ and $SupMed^t_{2,1}$ (Fig. 4, Sect. 4.2), with $t^{min}_{1,2}=t^{min}_{2,1}=1$ and $t^{max}_{1,2}=t^{max}_{2,1}=2$. Let's notice that the timers and counters, used for specifying a desired service and the supremal behaviour of the medium, are *fictitious*. For example, the desired service of Figure 3.b just means that the user wants that there must be at most two ticks between primitives A1 and B2. But the timers do not really exist.

Der_Seq_Prot (Sect. 7.2) is used and the intermediate results of Steps 1 to 8 are represented on Appendix B.

Step 9 : The specifications obtained are represented on Figure 16.
$PS^{tp}_1=(Q_1, V_1, \delta^{tp}_1, \mathcal{I}_1, \mathcal{T}_1, q_{1,0}, i_{1,0}, t_{1,0})$, with :
- $\mathcal{I}_1=\{1\}$, $\mathcal{T}_1=\{0,1,2,3\}$, $q_{1,0}=1$, $i_{1,0}=1$, $t_{1,0}=0$.
- The transformation functions are: $F_1(1,0)=F_1(1,1)=F_1(1,2)=(1,0)$, $F_2(1,0)=(1,0)$, $F_3(1,1)=(1,0)$ $F_4(1,*)=(1,1)$ and $F_5(1,*)=(1,2)$, where $*$ is any value $\mathcal{T}_1$.

$PS^{tp}_2=(Q_2, V_2, \delta^{tp}_2, \mathcal{I}_2, \mathcal{T}_2, q_{2,0}, i_{2,0}, t_{2,0})$
- $\mathcal{I}_2=\{1\}$, $\mathcal{T}_2=\{0,1,2\}$, $q_{2,0}=1$, $i_{2,0}=1$, $t_{2,0}=0$.
- The transformation functions are: $G_1(1,*)=(1,0)$, $G_2(1,*)=(1,1)$, $G_3(1,0)=G_3(1,1)=(1,0)$ $G_4(1,0)=(1,0)$ and $G_5(1,1)=(1,0)$, where $*$ is any value $\mathcal{T}_2$.

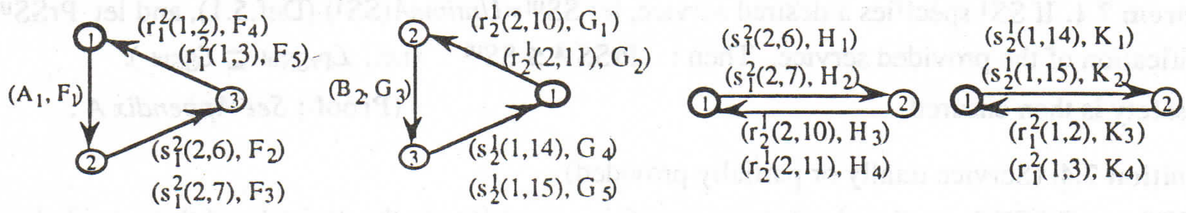
$ReqMed^{tp}_{1,2}(2)=(Q_3, V_3, \delta^{tp}_3, \mathcal{I}_3, \mathcal{T}_3, q_{3,0}, i_{3,0}, t_{3,0})$
- $\mathcal{I}_3=\{1,2\}$, $\mathcal{T}_3=\{0,1,2,3\}$, $q_{3,0}=1$, $i_{3,0}=1$, $t_{3,0}=0$.

- The transformation functions are: $H_1(1,*)=(1,0)$, $H_2(1,*)=(2,0)$, $H_3(1,1)=(1,0)$
$H_4(1,2)=H_4(2,1)=(1,0)$, where $*$ is any value $\mathcal{T}_3$.

$ReqMed^{tp}_{2,1}(1)=(Q_4, V_4, \delta^{tp}_4, \mathcal{I}_4, \mathcal{T}_4, q_{4,0}, i_{4,0}, t_{4,0})$

- $\mathcal{I}_4=\{1,2\}$, $\mathcal{T}_4=\{0,1,2,3\}$, $q_{4,0}=1$, $i_{4,0}=1$, $t_{4,0}=0$.
- The transformation functions are: $K_1(1,*)=(1,0)$, $K_2(1,*)=(2,0)$, $K_3(1,1)=(1,0)$
$K_4(1,2)=K_4(2,1)=(1,0)$, where $*$ is any value $\mathcal{T}_4$.

Let's mention that $\mathcal{I}_3=\mathcal{I}_4$, $\mathcal{T}_3=\mathcal{T}_4$, and $K_i=H_i$, for $i=1, 2,3,4$.
The elements of Q_i and V_i, for $i=1,2,3,4$, are represented on Figure 16.



16.a. PS_1^{tp} **16.b.** PS_2^{tp} **16.c.** $ReqMed^{tp}_{1,2}(2)$ **16.d.** $ReqMed^{tp}_{2,1}(1)$

Figure 16. Temporized specifications obtained at Step 9

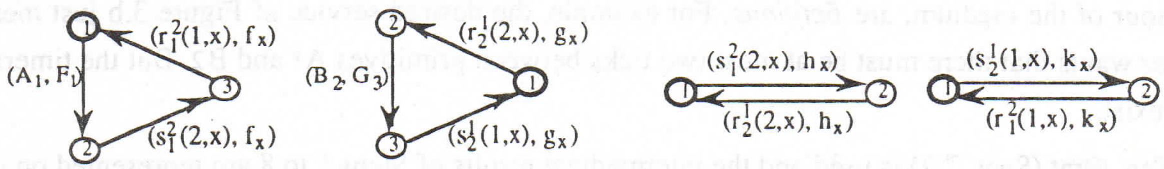
Step 10 : The specifications obtained are represented on Figure 17, with :

$f_2=F_4$, $f_3=F_5$, $f_6=F_2$, $f_7=F_3$;

$g_{10}=G_1$, $g_{11}=G_2$, $g_{14}=G_4$, $g_{15}=G_5$;

$h_6=H_1$, $h_7=H_2$, $h_{10}=H_3$, $h_{11}=H_4$;

$k_2=K_3$, $k_3=K_4$, $k_{14}=K_1$, $k_{15}=K_2$;



17.a. PS_1^{tp} **17.b.** PS_2^{tp} **17.c.** $ReqMed^{tp}_{1,2}(2)$ **17.d.** $ReqMed^{tp}_{2,1}(1)$

Figure 17. Parameterized temporized specifications obtained at Step 10

The informal semantics of PS_1^{tp}, PS_2^{tp}, $ReqMed^{tp}_{1,2}(2)$, and $ReqMed^{tp}_{2,1}(1)$ is the following. If PE_1 and PE_2 respect respectively the specifications PS_1^{tp} and PS_2^{tp}, and if the medium respects the specifications $ReqMed^{tp}_{1,2}(2)$ and $ReqMed^{tp}_{2,1}(1)$, then the desired service SS^t of Section 4.1 is *totally* provided.
The service is totally provided (Def. 7.4) because the projection of $MedSS^{ut}$ (Step 4) in alphabet $V\cup\{tick\}$, is equivalent to $SS^{ut}=UntimeA(SS^t)$.

8. Protocol derivation for parallel and concurrent real-time systems

8.1. Introduction

For the sake of simplicity and without a loss of generality, we consider only a parallel system composed by *two* sequential systems. A desired parallel service is then specified by two TA (Def.4.6) $SS^t[i]$ over alphabets $V[i]$, for $i=1,2$. Each $SS^t[i]$ specifies a sequential desired service. Let's consider three cases :

(a) $V[1] \subseteq V[2]$: $SS^t = SS^t[1] \otimes SS^t[2]$ (Def.4.15) is a sequential service (Remark 4.4.b), and we may use the procedure *Der_Seq_Prot* (Sect. 7.2) for deriving the protocol providing the service specified by SS^t.

(b) $V[i] \neq \emptyset$ and $V[i] \cap V[j] = \emptyset$, for i,j=1, 2, and i≠j : $SS^t[1]$ and $SS^t[2]$ are independent and compose a parallel system (Def. 4.11). We may process each sequential service separately, i.e., for each $SS^t[i]$, we use *Der_Seq_Prot* for deriving the sequential protocol which provide $SS^t[i]$.

(c) $V[i]-V[j] \neq \emptyset$ and $V[i] \cap V[j] \neq \emptyset$, for i,j=1, 2, and i≠j : $SS^t[1]$ and $SS^t[2]$ are dependent and compose a concurrent system (Def. 4.11). This case is studied in detail in the rest of the present section 8.

8.2. Solution for the problem of the choice

In a concurrent system, we think that one of the main problems consists in avoiding possible deadlocks. For that, Rule 2 (Sect. 7) is too weak. Therefore, a more restrictive rule is used.

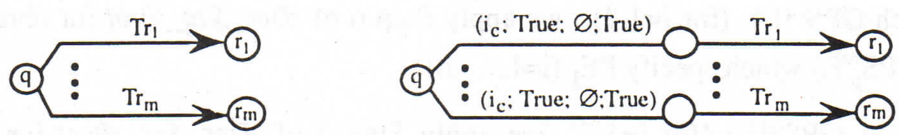
Rule 3. All choices are executed by a same protocol entity PE_c. Therefore, after execution of a primitive A_a by PE_a, the choice between several primitives executed by different PE_{bi}, for i=1,2,..., p, is achieved by PE_c as follows : - PE_a "passes the buck" to a given PE_c;

- PE_c selects PE_{bi} and the primitive to be executed. □

Rule 3 seems too restrictive, and we intend to weaken it in a next version.

To respect explicitly Rule 3, we must add to $SS^t[1]$ and $SS^t[2]$ some timed transitions (Def. 4.5) noted $[q_1, i_c, q_2, True, \emptyset, True]$, where i_c is executed by PE_c. These timed transitions are added as follows :

For each state q of $SS^t[i]$, for i=1, 2, where nbrout(q)>1 (Def. 7.1), the structure of Figure 18.a. is replaced by the structure of Figure 18.b, where Tr_1, ..., Tr_m are ougoing transitions of state q. The specifications obtained are noted $SS^t_c[1]$ and $SS^t_c[2]$.



18.a. Before adding events i_c **18.b.** After adding events i_c

Figure 18. Adding events i_c

Instead of using Rule 1 (Sect.7) for the two services $SS^t[1]$ and $SS^t[2]$, the following stronger rule is used.

Rule 4. The outgoing transitions (Def. 7.1) of the two initial states $q_{1,0}$ and $q_{2,0}$ of $SS^t[1]$ and $SS^t[2]$ are executable by *a same protocol entity*, i.e., $out(q_{1,0})=out(q_{2,0})$ and $|out(q_{1,0})|=|out(q_{2,0})|=1$. □

Informally, Rule 4 requires that the first action of the desired service is always executed by a same site. Without Rule 4, if $SS^t[1]$ and $SS^t[2]$ are dependent then synchronizing the local clocks of the different sites is mandatory, and the transformation of Step 8 of *Der_Seq_Prot* cannot be used.

8.3. Procedure of protocol derivation for a concurrent system

Let two TA $SS^t[i]$ over alphabets $V[i]$, for $i=1,2$, and a TA $SupMed^t_{u,v}$ for each pair (PE_u, PE_v), the procedure of protocol synthesis for concurrent systems, called *Der_Conc_Prot*, consists of eleven steps.

Step 1 : $SS^t[i]$ are modified into $SS^t_c[i]$, for $i=1,2$, (Sect. 8.2). Besides, any two states of respectively $SS^t_c[1]$ and $SS^t_c[2]$ must be identified differently. This is necessary for not confusing exchanged messages, which are parameterized by identifiers of states (see *Der_Seq_Prot* in Sect. 7.2).

Step 2 : Steps 1 to 5 of *Der_Seq_Prot* are applied to each $SS^t_c[i]$ for obtaining $GPS^{ut}_c[i]$, for $i=1,2$, but with the following difference : at the third step of *Der_Seq_Prot*, not only transitions ε, but also transitions executing event ic are removed. Let $V_g[i] \cup \{tick\}$ be the alphabet of $GPS^{ut}_c[i]$, then $V[i] \subseteq V_g[i]$.

Step 3 : The synchronized product $GPS^{ut}_c = GPS^{ut}_c[i] \times GPS^{ut}_c[i]$ is computed.

Step 4 : Indesirable states are removed from GPS^{ut}_c for obtaining GPS^{ut}. A state is indesirable if it is either a deadlock or only a selfloop *tick* is executable from it (Remark 5.1). For removing indesirable states, we may use a fixpoint method similar to the one used in the control theory for computing supremal controllable languages [12,32].

Step 5 : The untimed protocol specification PS^{ut}_c of PE_c (Sect. 8.2) is obtained by projecting GPS^{ut} in alphabet $V_c \cup \{tick\}$. V_c contains all events of GPS^{ut} executed by PE_c, and these events are of the form $s^*_c(*,*)$ and $r^*_c(*,*)$ (see Step 2 of *Der_Seq_Prot*), where $*$ may be any parameter.

Step 6 : The sequential $GPS^{ut}[i]$ are obtained by projecting GPS^{ut} in alphabets $V_g[i] \cup \{tick\}$ of $GPS^{ut}_c[i]$ (Step 2), for $i=1, 2$. The sequential processes specified by $GPS^{ut}[i]$, for $i=1,2$, interact with PE_c specified by PS^{ut}_c and do not lead to an indesirable state.

Step 7 : For each $GPS^{ut}[i]$ (for $i=1,2$), we apply Step 6 of *Der_Seq_Prot* for obtaining the untimed automata (UA) $PS^{ut}_j[i]$ which specify PE_j ($j=1, \ldots ,n$).

Step 8 : For each $GPS^{ut}[i]$ (for $i=1,2$), we apply Step 7 of *Der_Seq_Prot* for obtaining the UA $ReqMed^{ut}_{j,k}(q)$. Each $ReqMed^{ut}_{j,k}(q)$ depends implicitly on i, because q identifies a state of $SS^t_c[i]$, and states of $SS^t_c[1]$ and $SS^t_c[2]$ are identified differently (see Step 1).

The informal semantics of PS^{ut}_c (Step 5), $PS^{ut}_j[i]$ (Step 7), and $ReqMed^{ut}_{j,k}(q)$ (Step 8) is the following. If each PE_j, for $j=1, \ldots ,n$, is specified by $PS^{ut}_j[1] \times PS^{ut}_j[2]$, and if the medium respects the specifications $ReqMed^{ut}_{i,j}(q)$, then the desired concurrent service specified by $SS^t = SS^t[1] \times SS^t[2]$ is totally or partially provided by the help of PE_c specified by PS^{ut}_c.

Step 9 : Since Rule 4 is respected, the transformation of Step 8 of *Der_Seq_Prot* is applied to the untimed specifications obtained at Steps 6,7 and 8. With this transformation, the clocks of the different sites do not need to be synchronized.

Step 10 : The untimed specifications obtained at Step 9 are transformed into temporized automata (Sect.6, Step 9 of *Der_Seq_Prot*).

Step 11 : The transformation of Step 10 of *Der_Seq_Prot* is applied to the temporized specifications obtained at Step10, which becomes more concise.

End of *Der_Conc_Prot* ■

8.4. Example

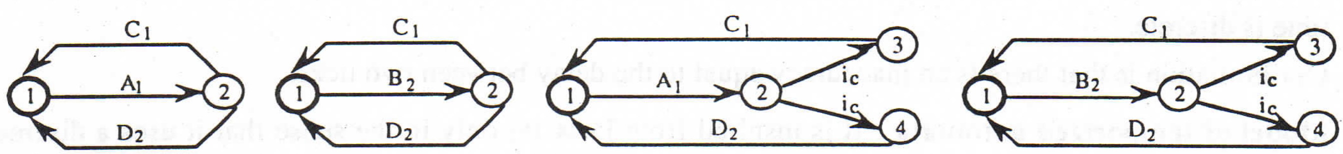
Since the problem of concurrency exists even for systems without timing requirements, let's give an example for such systems. In this case, the T_Conditions (Def.4.2) and F_Conditions (Def. 4.4) of timed transitions (Def.4.5) are True, and their Resets are ∅. The untiming operation (Def. 5.1) consists just in adding a selfloop *tick* to every state. For these reasons : (a) timed transitions $[q_1,A_i,q_2,True,\emptyset,True]$ are represented just by $[q_1,A_i,q_2]$; (b) event *tick* is not represented, therefore A^t, $A^{ut}=UntimeA(A^t)$, and $A^{tp}=Temp(A^{ut})$ are not differentiated and are refered to by A; (c) the messages exchanged contain only the first parameter. The second parameter which implicitly contains only temporal informations, is not necessary. Then :

(a) Step 2 of *Der_Conc_Prot* is composed only by steps 1 to 3 of *Der_Seq_Prot*.

(b) Step 4 of *Der_Conc_Prot* just consists in removing deadlocks.

(c) Steps 8 to 11 of *Der_Conc_Prot* are not necessary.

The desired concurrent service is represented on Figures 19.a and 19.b, and is specified by SS[1] and SS[2] respectively over alphabets $V[1]=\{A_1, B_2, \alpha_2\}$ and $V[2]=\{A_1, B_2, \gamma_2\}$. SS[1] and SS[2] are then synchronized on C_1 and D_2. After the first step, we obtain $SS_c[1]$ and $SS_c[2]$ on Figures 19.c. and 19.d.

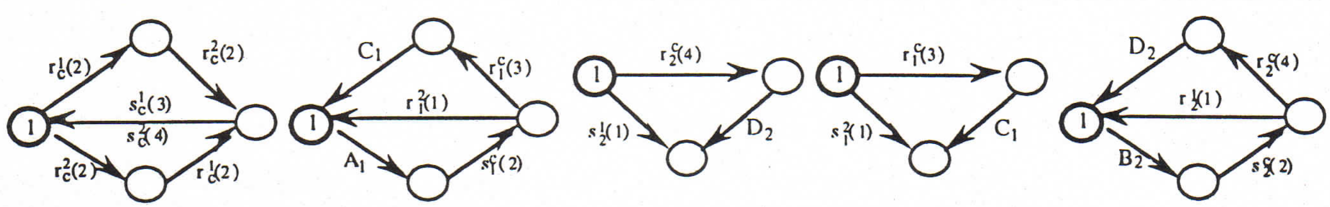


19.a. SS[1] **19.b.** SS[2] **19.c.** $SS_c[1]$ **19.d.** $SS_c[2]$

Figure 19. Example of concurrent desired service without timing requirements

If we apply *Der_Conc_Prot*, we obtain :

- at Step 5, the specification PS_c of PE_c is represented on Figure 20.a.
- at Step 7, the specifications $PS_1[1]$, $PS_2[1]$, $PS_1[2]$ and $PS_2[2]$, are represented on Figures 20.b to 20.e.



20.a. SS_c **20.b.** $PS_1[1]$ **20.c.** $PS_2[1]$ **20.d.** $PS_1[2]$ **20.e.** $PS_2[2]$

Figure 20. Specifications od the protocol entities

9. Conclusion

In this paper, we have developed two models for specifying real-time discrete event systems. The first model, based on timed automata, is used to specify a desired service and a supremal behaviour of the medium. The second model based on temporized automata, is used to model the protocol and temporal constraints on the medium. Next, the two models are applied to synthesize protocols for real-time applications. Two procedures of protocol synthesis, respectively for sequential and parallel distributed real-time discrete event systems, are proposed. The synthesis approach used for deriving a real-time protocol providing a desired service is inspired by other works, but our main contribution has been to consider *timing constraints* .

Let's make an informal and succint comparison between our two models and models which have mainly inspired us.

Model of timed automata : It is partially inspired from [1,31]. In [1], a *dense time* and a set of clocks are used. The clocks are used as we use the timers in our model, but the semantics is quite different because we use a discrete time. The main advantages of our model are :

(a) Contrary to our model, where the finiteness property can be ensured by using counters, the finiteness property is supposed respected in [1,31], but it is not ensured.

(b) In [31], the composition is defined only when the two TA have a same alphabet, and in [1], the authors specify only how events executed conjointly by the two composed systems are processed.

(c) Algorithms which deal with *distributed* real-time systems are relatively straightforward when the time is discrete.

Our limitation is that there is an inaccuracy equal to the delay between two ticks.

Model of temporized automata : It is inspired from [5,24,25] only in the sense that it uses a discrete time, and a fictitious event tick generated by a conceptual clock. Our model is more suitable to be automatically computed from an untimed automaton.

And to conclude, we propose the following future works. Firstly, we intend to replace Rule 3 (Sect.8.2) by a weaker rule. Secondly, we are investigating how we can modify systematically several existing protocol entities, which provide an old service, for providing a new desired service. For that, we intend to use control theory of the discrete event systems .

References

[1] R. ALur and D.Dill, "Automata for Modeling Real-Time Systems." In Lecture Notes in Computer Science 443, editor Proceedings of the 17th Intern. Coll. on Automata, Languages and Programming, Warwick, UK, 1990. Springer-Verlag.

[2] W.A.Barrett and J.D.Couch,"Compiler Construction:Theory and Practice" Ed.:Science Research Associates, Inc. 1979

[3] G.v. Bochmann and R. Gotzhein, "Deriving protocols specifications from service specifications." Proceedings du Symposium ACM SIGCOM' 86, Vermont, USA, pp.148-156, 1986.

[4] B. Berthomieu and M. Diaz, "Modeling and Verification of Time Dependent Systems Using Time Petri Nets." IEEE Transactions on Software Engineering, Vol.17, No.3, pp.259-273, March 1991.

[5] B. Brandin and W.M. Wonham, "The supervisory Control of Timed Discrete-Event Systems." Proceedings of the 31rst Conf. on Decision and Control, Tucson, Arizona, Dec.92.

[6] T.Y. Choi, "Sequence Method for Protocol Construction." 6th IFIP Int. Symp. on Protocol Specification, testing, and Verification, pp.307-321, May 1986.

[7] P.M. Chu and M.T. Liu, " Synthesizing Protocol Specifications From Service Specification in FSM Model." Proc. Computer Networking Symp., pp.173-182, April 1988.

[8] D. Dill, "Timing assumptions and Verifications of Finite-State Concurrent Systems.", In Lecture Notes in Computer Sciences 407, editor Automatic Verification Methods For Finite State Systems, Intern. Workshop, pp.197-212, Grenoble France, 1989. Springer-Verlag.

[9] M.G. Gouda and Y.T. Yu, "Synthesis of Communicating Finite State Machines with Garanteed Progress." IEEE Transactions on Communication, COM-32, No.7, pp.779-788, July 1984.

[10] A. Khoumsi, G.v. Bochmann and R. Dssouli, "Dérivation de spécifications de protocoles à partir de spécifications de services avec contraintes temporelles." Colloque Francophone pour l'ingénierie des protocoles (CFIP), Montreal, September 1993.

[11] A. Khoumsi, G.v. Bochmann and R. Dssouli, "On specifying services and synthesizing protocols for real-time applications." Conference on Protocol Specification, Testing and Verification (PSTV), Vancouver, June 1994.

[12] A. Khoumsi, G.v. Bochmann and R. Dssouli, "Contrôle et extension des systèmes à événements discrets totalement et partiellement observables." Third Maghrebian Conference on Software Engineering and Artificial Intelligence, Rabat, April 1994.

[13] F. Khendek, G.v. Bochmann and C. Kant, "New results on deriving protocol specifications from services specifications.", Proceedings of the ACM SIGCOMM'89, pp.136-145, 1989.

[14] C. Kant, T. Higashino and G.v. Bochmann, "Deriving protocol specifications from service specifications written in LOTOS." Rapport interne No 805, Département d'Informatique et de Recherche Opérationnelle. Faculté des arts et des sciences, Université de Montréal, January 1992.

[15] M. Kapus-Kolar, "Deriving Protocol Specifications from Service Specifications with Heterogeneous Timing Requirements." Proc. IEEE Int. Conf. on Software Engineering for real time systems, United-Kingdom, 1991.

[16] M. Kapus-Kolar and J. Rugelj, "Deriving Protocol Specifications from Service Specifications with Simple Relative Timing Requirements." Proc. ISMM Int. Workshop on parallel computing, Italy 1991.

[17] Y. Kakuda and Y.Wakahara, "Component-based Synthesis of Protocols for Unlimited Number of Processes." Proc. COMPSAC'87, pp.721-730, 1988.

[18] G. Leduc and L. Léonard, "A Timed LOTOS supporting a dense time domain and including new timed operators." In: M. Diaz, R. Groz, eds., FORTE'92, Perros-Guirec, France, Oct. 1992 (North-Holland, Amsterdam, 1993).

[19] J-P. Courtiat, M. S. de Camargo, D-E. Saidouni, " RT-LOTOS: LOTOS temporisé pour la spécification de systèmes temps réel." Colloque Francophone pour l'ingénierie des protocoles (CFIP), Montreal, September 1993.

[20] P. Merlin, "A Study of the recoverability of computer system." Thesis, Dep. Comput. Sci., California, Irvine, 1974.

[21] P. Merlin and D.J. Faber, "Recoverability of communication protocols." IEEE Transactions on Communication, Vol. COM-24, No.9, September 1976.

[22] P. Merlin and G.v.Bochmann, "On the Construction of Submodule Specifications and Communication Protocols." ACM Transactions on Programming Languages and Systems, No.1, pp.1-25, January 1983.

[23] N. Rico, G.v. Bochmann and O. Cherkaoui, "Model-checking for real-time systems specified in Lotos." Conference on Computer Aided Verification (CAV), pp-277-288, 1992.

[24] J.S. Ostroff, "Deciding Properties of Timed Transitions Models." IEEE Transactions on Parallel and Distributed Systems, Vol.1, No.2, pp.170-183, April 1990.

[25] J.S. Ostroff and W.M. Wonham, "A framework for real-time discrete event control." IEEE Transactions on Automatic Control, Vol.35, No.4, pp.386-397, April 1990.

[26] C. Ramchandani, "Analysis of Asynchronous Concurrent Systems by Timed Petri Nets." Massachussets Inst. Technol., Project MAC, Tech. Rep. 120, February 1974.

[27] C.V. Ramamorthy, S.T. Dong and Y. Usuda, "An Implementation of an Automated Protocol Synthesizer (APS) and its Application to the X.21 Protocol." IEEE Transactions on Software Engineering, Vol.SE-11(9), pp.886-908, September 1985.

[28] D.P. Sidhu, "Rules for Synthesizing Correct Communication Protocols.", ACM SIGCOM comput. Commun., rev. Vol.12(1), pp.35-51, January 1982.

[28] D.P. Sidhu, "Protocol Design Rules." 2nd IFIP Int. Symp. on Protocol Specification, testing, and Verification, pp.283-300, 1982.

[29] K.Saleh and R. Probert, "A service-based method for the synthesis of Communications protocols." International Journal of Mini and Microcomputers, Vol.12, No 3, 1990.

[30] H.W-Toi and Gérard Hoffmann, "The Control of Dense Real-Time Discrete Event Systems." Report submitted to IEEE Transactions on Automatic Control, February 1992.

[31] W.M. Wonham and P.J. Ramadge, "On the Supremal Controllable sublanguage of a Given Language." SIAM J.Control and Optimization, Vol.25, No.3, May 1987.

[32] M.C. Yuang, "Survey of Protocol Verification Techniques Based on Finite State Machine Models." Proc. of Computer Networking Symp., pp.164-172, April 1988.

[33] Y.X. Zhang, K. Takahashi, N. Shiratori and S. Noguchi, "An interactive Protocol Synthesis Algorithm Using a Global State Transition Graph." IEEE Transactions on Software Engineering, SE-14(3), pp.394-404.

[33] P. Zafiropulo, C. West, H. Rudin, D. Cowan and D. Brand, "Towards Analyzing and Synthesizing Protocols." IEEE Transactions on Communication, Vol. COM-28, No.4, pp.651-661, April 1980.

Table of Figures

APPENDIX A : Proofs

Proof of Property 3.1.

Let TRC= $\alpha_1\alpha_2 \ldots \alpha_j \ldots$ be a infinite untimed trace respecting the finiteness property (FP), and Trc=$TimeT$(TRC)=$\langle\sigma_1,\tau_1\rangle\ldots\langle\sigma_i,\tau_i\rangle\ldots$. Since TRC respects the FP, then Trc also respects the FP (Def. 3.5).

Since Trc is infinite, then $\forall\ i > 0$, σ_i and σ_i are defined (1)

Def. 3.1 implies : $\exists$ Mc > 0, such that : $\forall\ i > 0$, $\exists\ j > i$ with $\tau_{j-1} = \tau_i < \tau_j$ and $j \leq i+Mc$. (2)

Def. 3.5 implies : ($\alpha_{i+\tau_i}=\sigma_i$) and ($\alpha_k=tick$, if $\nexists$ j>0 such that $k=j+\tau_j$), for i,j=1,2, ... (3)

Let Mc defined in (2), k>0, i such that $i+\tau_i \leq k < i+1+\tau_{i+1}$; (4)

- $l_1 = (i+1)+\tau_{i+1} > k \geq i+\tau_i$: From (3), $\alpha_{l_1} = \sigma_{i+1} \neq tick$;

Let's consider two cases, for computing l_2:

Case 1: $k+1 < i+1+\tau_{i+1}$; (5)
- $l_2 = k+1 > k$; (6)
- (4), (5) and (6) imply : $i+\tau_i < l_2 < i+1+\tau_{i+1}$; (7)
- (6) implies : $l_2 - k = 1 < Mc+1$;
- (3) and (7) imply : $\alpha_{l_2} = tick$;

Case 2: $k+1 = i+1+\tau_{i+1}$; (8)
- (2) implies : $\exists$ j such that $i+1 < j \leq i+1+Mc$ and $\tau_{j-1} = \tau_{i+1} < \tau_j$; (9)
- $l_2 = j+\tau_{i+1}$; (10)
- (8) and (10) imply : $l_2 - k = j - i$; (11)
- (9) and (10) imply : $j-1+\tau_{j-1} < l_2 < j+\tau_j$; (12)
- (3) and (12) imply : $\alpha_{l_2} = tick$;
- (9) and (11) imply : $l_2 > k$ and $l_2 - k \leq Mc+1$; ■

Proof of Theorem 3.1.

To demonstrate the equality between $UntimeL(\mathcal{L}_1 \cap \mathcal{L}_2)$ and $UntimeL(\mathcal{L}_2) \cap UntimeL(\mathcal{L}_2)$, we will prove the inclusions in the two directions.

(1) $UntimeL(\mathcal{L}_1 \cap \mathcal{L}_2) \subset UntimeL(\mathcal{L}_1) \cap UntimeL(\mathcal{L}_2)$

Let TRC $\in UntimeL(\mathcal{L}_1 \cap \mathcal{L}_2)$.

Def.3.6 implies : $\exists$ Trc $\in \mathcal{L}_1 \cap \mathcal{L}_2$ such that TRC=$UntimeT$(Trc) and Trc=$TimeT$(TRC) (1)

(1) implies : Trc $\in \mathcal{L}_1$, Trc $\in \mathcal{L}_2$, and TRC=$UntimeT$(Trc) (2)

Def. 3.6 and (2) imply : TRC $\in UntimeL(\mathcal{L}_1)$ and TRC $\in UntimeL(\mathcal{L}_2)$ (3)

Therefore, (3) implies : TRC $\in UntimeL(\mathcal{L}_1) \cap UntimeL(\mathcal{L}_2)$

(2) $UntimeL(\mathcal{L}_1) \cap UntimeL(\mathcal{L}_2) \subset UntimeL(\mathcal{L}_1 \cap \mathcal{L}_2)$

Let TRC $\in UntimeL(\mathcal{L}_1) \cap UntimeL(\mathcal{L}_2)$ (4)

(4) implies : TRC $\in UntimeL(\mathcal{L}_1)$ and TRC $\in UntimeL(\mathcal{L}_1)$ (5)

Def. 3.6 and (5) imply : $\exists$ $Trc_1 \in \mathcal{L}_1$ such that TRC=$UntimeT$(Trc_1) (6)

$\exists$ $Trc_2 \in \mathcal{L}_2$ such that TRC=$UntimeT$(Trc_2) (7)

Def. 3.5, (6) and (7) imply : Trc_1=$TimeT$(TRC) and Trc_2=$TimeT$(TRC) (8)

(6), (7) and (8) imply : $Trc_1 = Trc_2$=$TimeT$(TRC) $\in \mathcal{L}_1 \cap \mathcal{L}_2$ (9)

Therefore, (9) implies : TRC $\in UntimeL(\mathcal{L}_1 \cap \mathcal{L}_2)$ ■

Proof of Property 4.1.

Let : - $A^t=(Q,V,T,\mathcal{V},\delta,q_0)$, with $\mathcal{V}=\{Vc_1, Vc_2,...,Vc_{Nc}\}$;

- $Trc=\langle\sigma_1,\tau_1\rangle...\langle\sigma_i,\tau_i\rangle...$ be any timed trace accepted by A^t;
- $\mathcal{Tr}=Tr_1 Tr_2...Tr_i...$ be the infinite sequence of transitions of A^t, which accepts Trc, with $Tr_i=[q_{i-1};\sigma_i;q_i;E_i(ts);R_i;K_i(cs)]\in\delta$, for $i>0$;
- $Mc = Mc_1+Mc_2+...+Mc_{Nc}$.

We intend to prove that : ($Vc_1\cup...\cup Vc_{Nc}=V$) implies : ($\forall\ i>0$, $\exists j>i$ with $\tau_{j-1} = \tau_i < \tau_j$ and $j \leq i+Mc$).

Let then $i > 0$ and $j=i+Mc$. Let's prove that $\tau_i < \tau_j$ or, in other words, that $\tau_i = \tau_j$ is impossible.

Hypotheses : $\tau_i = \tau_j$: (1)

$Vc_1\cup...\cup Vc_{Nc}=V$ (2)

(1) implies : there is no tick between the occurrences of Tr_i and Tr_j. (3)

(3) implies : no counter is set to zero between occurrences of Tr_i and Tr_j. (4)

Def. 4.5 and (2) imply : The F_Condition of any Tr_i depends on at least one counter, i.e., $K_i(cs)$ is not the constant True. (5)

(5) implies : at least one counter is incremented with the occurrence of any transition. (6)

Def. 4.9 implies : no counter c_p can be incremented more than Mc_p times without being reset to zero (7)

j=i+Mc implies : there are Mc+1 transitions from Tr_i to Tr_j. (8)

(4), (6), (7) and (8) imply : $Mc_1+Mc_2+...+Mc_{Nc} \geq Mc+1$ (9)

Therefore, hypotheses (1) and (2) lead to inequation (9) which is incompatible with $Mc = Mc_1+Mc_2+...+Mc_{Nc}$.

We deduce that hypothesis (2) implies $\tau_i < \tau_j$ and then Trc respects the finiteness property. ■

Proof of Theorem 4.1.

Let $A^t_k=(Qk,V,Tk,\mathcal{V}k,\delta k,qk_0)$ be two TA, for k=1,2, over a same alphabet V. To demonstrate the equality between $\mathcal{L}_{A^t_1\times A^t_2}$ and $\mathcal{L}_{A^t_1}\cap\mathcal{L}_{A^t_2}$, we will prove the inclusions in the two directions.

(1) $\mathcal{L}_{A^t_1}\cap\mathcal{L}_{A^t_2}\subset\mathcal{L}_{A^t_1\times A^t_2}$

Let $Trc=\langle\sigma_1,\tau_1\rangle...\langle\sigma_i,\tau_i\rangle...\in\mathcal{L}_{A^t_1}\cap\mathcal{L}_{A^t_2}$, and then $Trc\in\mathcal{L}_{A^t_1}$ and $Trc\in\mathcal{L}_{A^t_2}$. (1)

(1) and Def. 4.10 imply that there exists, for k=1, 2, an infinite sequence $\mathcal{Tr}k=Trk_1 Trk_2...Trk_i...$ of transitions of A^t_k, which accepts Trc, with

$Trk_i=[qk_{i-1};\sigma_i;qk_i;Ek_i(tsk);Rk_i;Kk_i(csk)]\in\delta k$ [1], for $i>0$. (2)

(2) and Def. 4. 10 imply that any Trk_i is enabled at time τ_i, for $i>0$, and k=1, 2. (3)

(3) and Def. 4. 12 imply that transitions Tr_i of $A^t_1\times A^t_2$, are enabled at time τ_i, for $i > 0$, with $Tr_i=[\langle q1_{i-1}, q2_{i-1}\rangle; \sigma_i; \langle q1_i, q2_i\rangle; E1_i(ts1)\wedge E2_i(ts2); R1_i\cup R2_i); K1_i(cs1)\wedge K2_i(cs2)]$ [2]. (4)

(4) and Def. 4. 10 imply that $\mathcal{Tr}=Tr_1 Tr_2...Tr_i...$ accepts Trc. (5)

(5) and Def. 4. 10 imply that $A^t_1\times A^t_2$ accepts Trc, i.e., $Trc\in\mathcal{L}_{A^t_1\times A^t_2}$.

(2) $\mathcal{L}_{A^t_1\times A^t_2}\subset\mathcal{L}_{A^t_1}\cap\mathcal{L}_{A^t_2}$

Let $Trc=\langle\sigma_1,\tau_1\rangle...\langle\sigma_i,\tau_i\rangle...$ be a timed trace accepted by $A^t_1\times A^t_2$, i.e., $Trc\in\mathcal{L}_{A^t_1\times A^t_2}$. (6)

(6) and Def. 4. 10 imply that there exists an infinite sequence $\mathcal{Tr}=Tr_1 Tr_2...Tr_i...$ of transitions of $A^t_1\times A^t_2$, which accepts Trc, with :

$Tr_i=[\langle q1_{i-1}, q2_{i-1}\rangle; \sigma_i; \langle q1_i, q2_i\rangle; E1_i(ts1)\wedge E2_i(ts2); R1_i\cup R2_i); K1_i(cs1)\wedge K2_i(cs2)]$ [2]. (7)

(7) and Def. 4.10 imply that any Tr_i is enabled at time τ_i, for $i>0$. (8)

(8) implies that any transition Trk_i of A^t_k is enabled at time τ_i, with :

$Trk_i=[qk_{i-1};\sigma_i;qk_i;Ek_i(tsk);Rk_i;Kk_i(csk)]\in\delta k$ [1], for $i>0$, and k=1, 2. (9)

(9) implies that the infinite sequence $\mathcal{T}rk=\mathrm{Trk}_1\mathrm{Trk}_2...\mathrm{Trk}_i...$ of transitions of A^t_k accepts Trc, with $\mathrm{Trk}_i=[qk_{i-1};\sigma_i;qk_i;Ek_i(tsk);Rk_i;Kk_i(csk)]\in\delta k$ [1], for i>0, and k=1,2. (10)

(10) and Def. 4. 10 imply that Trc is accepted by A^t_1 and A^t_2, i.e., $\mathrm{Trc}\in\mathcal{L}_{A^t_1}\cap\mathcal{L}_{A^t_2}$.

Footnotes (Theorem 4.1).

[1] $Ek_i(tsk)$ is a T_Condition (Def.4.2), w.r.t. Tk, and $Rk_i\subseteq Tk$, for k=1,2,

[2] $E1_i(ts1)\wedge E2_i(ts2)$ is a conjunction of two T_Conditions, respectively w.r.t. T1 and T2.

Therefore $E1_i(ts1)\wedge E2_i(ts2)$ is a T_Condition, w.r.t. $T1\cup T2$.

$R1_i\cup R2_i)\subseteq T1\cup T2$.

$K1_i(cs1)\wedge K2_i(cs2)$ is a conjunction of two F_Conditions, respectively w.r.t. C1 and C2.

Therefore $K1_i(cs1)\wedge K2_i(cs2)$ is a T_Condition, w.r.t. $C1\cup C2$. ■

Proof of Property 4.2.

Property 4.1 and $Vc1_1\cup...\cup Vc1_{Nc1}=Vc2_1\cup...\cup Vc2_{Nc2}=V$ imply that both $\mathcal{L}_{A^t_1}$ and $\mathcal{L}_{A^t_2}$ respect the finiteness property (FP). Therefore, their intersection $\mathcal{L}_{A^t_1}\cap\mathcal{L}_{A^t_2}$ also respects the FP. (1)

Theorem 4.1 and (1) implies $\mathcal{L}_{A^t_1\times A^t_2}=\mathcal{L}_{A^t_1}\cap\mathcal{L}_{A^t_2}$ and then $\mathcal{L}_{A^t_1\times A^t_2}$ respects the FP. (2)

Since $\mathcal{L}_{A^t_1}$, $\mathcal{L}_{A^t_2}$ and $\mathcal{L}_{A^t_1\times A^t_2}$ respect the FP, then A^t_1, A^t_2, and $A^t_1\times A^t_2$ also respect the FP (Prop. 4.1) ■

Proof of Lemme 4.1.

Let - A^t be a TA over V, and let W be such that $V\subseteq W$.

- $\mathrm{Trce}=\langle\sigma_1,\tau_1\rangle...\langle\sigma_i,\tau_i\rangle...$ be a timed trace over the alphabet W.

Def. 3.3, 4.10 and 4.14 imply :

$\exists\,\mathcal{T}r1$ which accepts $\mathrm{Proj}_V(\mathrm{Trce})$ and is a sequence of transitions of A^t $\Leftrightarrow$

$\exists\,\mathcal{T}r2$ which accepts Trce and is a sequence of transitions of $\mathrm{Ext}_W(A^t)$ (1)

(1) and Def. 4.10 imply : $\mathrm{Proj}_V(\mathrm{Trce})\in\mathcal{L}_{A^t}$ $\Leftrightarrow$ $\mathrm{Trce}\in\mathcal{L}_{\mathrm{Ext}_W(A^t)}$ (2)

Def. 3.4 implies : $\mathrm{Trce}\in\mathrm{Ext}_W(\mathcal{L}_{A^t})\Leftrightarrow(\mathrm{Proj}_V(\mathrm{Trce})\in\mathcal{L}_{A^t})$ (3)

(2) and (3) imply : $\mathrm{Trce}\in\mathcal{L}_{\mathrm{Ext}_W(A^t)}\Leftrightarrow\mathrm{Trce}\in\mathrm{Ext}_W(\mathcal{L}_{A^t})$, i.e.,

$$\mathcal{L}_{\mathrm{Ext}_W(A^t)}=\mathrm{Ext}_W(\mathcal{L}_{A^t})$$ ■

Proof of Theorem 4.2.

Let $A^t_i=(Qi,Vi,Ti,\mathcal{V}i,\delta i,qi_0)$, for i=1,2,

Def. 4.15 implies : $A^t_1\otimes A^t_2=\mathrm{Ext}_{V2}(A^t_1)\times A^t_2$, i.e., $\mathcal{L}_{A^t_1\otimes A^t_2}=\mathcal{L}_{\mathrm{Ext}_{V2}(A^t_1)\times A^t_2}$ (1)

Theorem 4.1 implies : $\mathcal{L}_{\mathrm{Ext}_{V2}(A^t_1)\times A^t_2}=\mathcal{L}_{\mathrm{Ext}_{V2}(A^t_1)}\cap\mathcal{L}_{A^t_2}$ (2)

(1) and (2) imply : $\mathcal{L}_{A^t_1\otimes A^t_2}=\mathcal{L}_{\mathrm{Ext}_{V2}(A^t_1)}\cap A^t_1$ ■

Proof of Property 4.3.

Let $A^t_i=(Qi,Vi,Ti,\mathcal{V}i,\delta i,qi_0)$, for i=1,2.

Hypothesis : A^t_2 (and then $\mathcal{L}_{A^t_2}$) respects the finiteness property (FP). (1)

Theorem 4.2 implies : $\mathcal{L}_{A^t_1\otimes A^t_2}=(\mathcal{L}_{\mathrm{Ext}_{V2}(A^t_1)}\cap\mathcal{L}_{A^t_2})$ and then $\mathcal{L}_{A^t_1\otimes A^t_2}\subseteq\mathcal{L}_{A^t_2}$ (2)

(1) and (2) imply : $\mathcal{L}_{A^t_1\otimes A^t_2}$ (and then $A^t_1\otimes A^t_2$) respects the FP. ■

Proof of Theorem 4.3.

Def. 4.16 implies : $A_1^t \| A_2^t = Ext_{V1\cup V2}(A_1^t) \times Ext_{V2\cup V1}(A_2^t)$, i.e., $\mathcal{L}_{A_1^t \| A_2^t} = \mathcal{L}_{Ext_{V1\cup V2}(A_1^t) \times Ext_{V1\cup V2}(A_2^t)}$ (1)

Theorem 4.1 implies : $\mathcal{L}_{Ext_{V1\cup V2}(A_1^t) \times Ext_{V1\cup V2}(A_2^t)} = \mathcal{L}_{Ext_{V1\cup V2}(A_1^t)} \cap \mathcal{L}_{Ext_{V1\cup V2}(A_2^t)}$ (2)

(1) and (2) imply : $\mathcal{L}_{A_1^t \| A_2^t} = \mathcal{L}_{Ext_{V1\cup V2}(A_1^t)} \cap \mathcal{L}_{Ext_{V1\cup V2}(A_2^t)}$ ■

Proof of Property 4.4.

Let Trc be a timed trace accepted by $A_1^t \| A_2^t$ where A_1^t and A_2^t are two TA respecting the FP

Theorem 4.3 implies : Trc is accepted by both $Ext_{V1\cup V2}(A_1^t)$ and $Ext_{V2\cup V1}(A_2^t)$ (1)

(1) and Lemma 4.1 imply : Trc belongs to both $Ext_{V1\cup V2}(\mathcal{L}_{A_1^t})$ and $Ext_{V1\cup V2}(\mathcal{L}_{A_2^t})$ (2)

Def. 3.4 and (2) imply : $Proj_{V1}(Trc) \in \mathcal{L}_{A_1^t}$ and $Proj_{V2}(Trc) \in \mathcal{L}_{A_2^t}$ (3)

Since A_1^t and A_2^t respect the FP, then (3) implies : both $Proj_{V1}(Trc)$ and $Proj_{V2}(Trc)$ respect FP (4)

(4) implies : The number of events of V1 which occur in Trc during one uct is finite and bounded by a constant Mc1 (Def. 3.1). (5)

The number of events of V2 which occur in Trc during one uct is finite and bounded by a constant Mc2. (6)

(5) and (6) imply : The number of events of $V1\cup V2$ which occur in Trc during one uct is finite and bounded by a constant $Mc \leq Mc1 + Mc2$.

Therefore, $A_1^t \| A_2^t$ respects finiteness property. ■

Proofs of Lemmes 5.1.

Let $A^t = (Q, V, T, \mathcal{V}, \delta, q_0))$ be a TA, and let $A^{ut} = \mathit{UntimeA}(A^t)$.

5.1.a. A state of A^{ut} is defined by $\langle q, ts, cs\rangle$, where :

$q \in Q$, $ts = (t_1, \ldots, t_{Nt}) \in \mathcal{T} \subseteq \langle 0; Mt_1+1\rangle \times \ldots \times \langle 0; Mt_{Nt}+1\rangle$ and $cs \in \mathcal{C} \subseteq \langle 0; Mc_1\rangle \times \ldots \times \langle 0; Mc_{Nc}\rangle$

Therefore, $Q^{ut} \subseteq Q \times \mathcal{T} \times \mathcal{C}$, which implies that $|Q^{ut}| \leq |Q| \times |\mathcal{T}| \times |\mathcal{C}|$ (1)

Since $\mathcal{T} \subseteq \langle 0; Mt_1+1\rangle \times \ldots \times \langle 0; Mt_{Nt}+1\rangle$, then $|\mathcal{T}| \leq \prod_{i=1}^{Nt} (Mt_i+2) \leq (Mt+2)^{Nt}$ (2)

Since $\mathcal{C} \subseteq \langle 0; Mc_1\rangle \times \ldots \times \langle 0; Mc_{Nc}\rangle$, then $|\mathcal{C}| \leq \prod_{i=1}^{Nc} (Mc_i+1) \leq (Mc+2)^{Nc}$ (3)

(1), (2) and (3) imply : $|Q^{ut}| \leq |Q| * (Mt+2)^{Nt} * (Mc+1)^{Nc}$ ■

5.1.b. If q is a state of A^t and $\langle q, ts, cs\rangle$ is a state of A^{ut}, then for every event executable in A^t from q, there is at most one event ($\neq$tick) executable in A^{ut} from $\langle q, ts, cs\rangle$.

Seeing that ts can have at most $(Mt+2)^{Nt}$ different states, and that cs can have at most $(Mc+1)^{Nc}$ different states, then the number of transitions of A^{ut} not equal to *tick* is bounded by :

$$|\delta_1| = |\delta| * (Mt+2)^{Nt} * (Mc+1)^{Nc}. \quad (1)$$

From each state of A^{ut}, there is at most one *tick*, the number of transitions *tick* is then bounded by :

$$|\delta_2| = |Q| * (Mt+2)^{Nt} * (Mc+1)^{Nc}. \quad (2)$$

(1) and (2) imply : the number of transitions in A^{ut}, is bounded by

$$|\delta_1| + |\delta_2| = (|Q| + |\delta|) * (Mt+2)^{Nt} * (Mc+1)^{Nc}.$$ ■

5.1.c.

- For a state $q^{ut} = \langle q, ts, cs\rangle$ of A^{ut}, the calculation of $\langle q, ts+1, 0\rangle$ necessitates a time in $O(Nt * \log_2(Mt+1) + Nc * \log_2(Mc))$, because $\log_2(Mt+1)$ (resp. $\log_2(Mc)$) is the maximum number of bits for coding the value of one timer (resp. one counter).

  Therefore, the time for calculating all transitions *tick* in A^{ut} is in :

$$O(|Q^{ut}| * \{Nt * \log_2(Mt+1) + Nc * \log_2(Mc)\}) \leq O(|Q| * (Mt+2)^{Nt} * (Mc+1)^{Nc} * \{Nt * \log_2(Mt+1) + Nc * \log_2(Mc)\})$$

– Testing if an event $\sigma \neq tick$ is enabled from a state $\langle q,ts,cs \rangle$ of A^{ut} necessitates at most a time in $O(k*(\log_2(Mt+1)+\log_2(Mc)))$ where k is the maximum length of the T_Conditions and the F_Conditions.

Calculating $\langle q',ts',cs' \rangle$ such that $[\langle q, ts, cs \rangle;\sigma;\langle q',ts',cs' \rangle] \in \delta^{ut}$, i.e., resetting some timers and possibly incrementing some counters, necessitates at most a time in $O(Nt*\log_2(Mt+1)+Nc*\log_2(Mc))$.

Therefore the time for calculating all transitions $\neq tick$ in A^{ut} is in :

$$O(|\delta_1|*\{(Nt+k)*\log_2(Mt+1)+(Nc+k)*\log_2(Mc)\}) \leq$$
$$O(|\delta|*(Mt+2)^{Nt}*(Mc+1)^{Nc}*\{(Nt+k)*\log_2(Mt+1)+(Nc+k)*\log_2(Mc)\})$$

– Indesirable states (i.e., deadlock states and states from which only a selfloop *tick* is excutable) are removed from A^{ut}. For that, we mat use a fixpoint method similar to the one used in the control theory for computing controllable languages. The complexity of such a method is in :

$$O(|Q^{ut}|^2)=O(|Q|^2*(Mt+2)^{2\times Nt}*(Mc+1)^{2\times Nc}).$$

Therefore, the total complexity is in : $O(|Q|^2*(Mt+2)^{2\times Nt}*(Mc+1)^{2\times Nc})$. ■

Proofs of Properties 5.1.

5.1.a. Let $TRC=\alpha_1\alpha_2 \ldots \alpha_j \ldots$be an untimed trace accepted by $UntimeA(A_1^t \times A_2^t)$.

Def. 3.6 and 5.1 imply : There exists a timed trace Trc accepted by $A_1^t \times A_2^t$, i.e., $Trc \in \mathcal{L}_{A_1^t \times A_2^t}$, such that : $TRC = UntimeT(Trc)$ (1)

(1) and Theorem 4.1 imply : $Trc \in \mathcal{L}_{A_1^t} \cap \mathcal{L}_{A_2^t}$, and then $Trc \in \mathcal{L}_{A_1^t}$ and $Trc \in \mathcal{L}_{A_2^t}$ (2)

(1), (2), and Def. 3.6 imply :

$TRC \in UntimeL(\mathcal{L}_{A_1^t}) = \mathcal{L}_{A_1^{ut}}$ and $TRC \in UntimeL(\mathcal{L}_{A_2^t}) = \mathcal{L}_{A_2^{ut}}$, i.e., $TRC \in \mathcal{L}_{A_1^{ut}} \cap \mathcal{L}_{A_2^{ut}}$ (3)

Since A_1^{ut} and A_2^{ut} are FSMs, then : $\mathcal{L}_{A_1^{ut}} \cap \mathcal{L}_{A_2^{ut}} \subseteq \mathcal{L}_{A_1^{ut} \times A_2^{ut}}$ (4)

We have not the equality, because $\mathcal{L}_{A_1^{ut}}$ and $\mathcal{L}_{A_2^{ut}}$ contain only infinite traces, while $\mathcal{L}_{A_1^{ut} \times A_2^{ut}}$ may contain finite traces, if $A_1^{ut} \times A_2^{ut}$ contains deadlocks.

(3) and (4) imply : $TRC \in \mathcal{L}_{A_1^{ut} \times A_2^{ut}}$, i.e., TRC is accepted by $A_1^{ut} \times A_2^{ut}$

Therefore : $UntimeA(A_1^t \times A_2^t) \leq A_1^{ut} \times A_2^{ut}$ ■

5.1.b. Def. 4.15 and Property 5.1.a imply :

$UntimeA(A_1^t \otimes A_2^t) = UntimeA(Ext_{V2}(A_1^t) \times A_2^t) \leq UntimeA(Ext_{V2}(A_1^t)) \times UntimeA(A_2^t)$ (1)

Let $Ext_{V2}(UntimeA(A_1^t))$ obtained by adding selfloops of events of V2-V1, to each state of $UntimeA(A_1^t)$. Therefore : $UntimeA(Ext_{V2}(A_1^t) \leq Ext_{V2}(UntimeA(A_1^t))$ and (2)

$Ext_{V2}(UntimeA(A_1^t)) \times UntimeA(A_2^t) = UntimeA(A_1^t) \times UntimeA(A_2^t)$ (3)

(2) implies : $UntimeA(Ext_{V2}(A_1^t)) \times UntimeA(A_2^t) \leq Ext_{V2}(UntimeA(A_1^t)) \times UntimeA(A_2^t)$ (4)

(1) and (4) imply : $UntimeA(A_1^t \otimes A_2^t) \leq Ext_{V2}(UntimeA(A_1^t)) \times UntimeA(A_2^t)$ (5)

(3) and (5) imply : $UntimeA(A_1^t \otimes A_2^t) \leq UntimeA(A_1^t) \times UntimeA(A_2^t)$ ■

5.1.c. Let $V=V1 \cup V2$

Def. 4.16 and Property 5.1.a imply :

$UntimeA(A_1^t \| A_2^t) = UntimeA(Ext_V(A_1^t) \times Ext_V(A_2^t)) \leq UntimeA(Ext_V(A_1^t)) \times UntimeA(Ext_V(A_2^t))$ (1)

By definition of $Ext_V(UntimeA(A_i^t))$: (see proof of Property 5.1.b)

$UntimeA(Ext_V(A_i^t)) \leq Ext_V(UntimeA(A_i^t))$ and (2)

$Ext_V(UntimeA(A_1^t)) \times Ext_V(UntimeA(A_2^t)) = UntimeA(A_1^t) \times UntimeA(A_2^t)$ (3)

(2) implies :

$UntimeA\,(Ext_V(A^t_1)) \times UntimeA\,(Ext_V(A^t_2)) \leq Ext_V(UntimeA\,(A^t_1)) \times Ext_V(UntimeA\,(A^t_2))$ (4)

(1) and (4) imply : $UntimeA(A^t_1 \| A^t_2) \leq Ext_V(UntimeA\,(A^t_1)) \times Ext_V(UntimeA\,(A^t_2))$ (5)

(3) and (5) imply : $UntimeA(A^t_1 \| A^t_2) \leq UntimeA(A^t_1) \times UntimeA\,(A^t_2)$ ■

5.1.d. Let $\mathcal{L}_{A^t_1}$ and $\mathcal{L}_{A^t_2}$ be such that $\mathcal{L}_{A^t_1} \subseteq \mathcal{L}_{A^t_2}$ (1)

Def. 5.1 implies : $\mathcal{L}_{UntimeA(A^t_1)} = UntimeL(\mathcal{L}_{A^t_1})$ and $\mathcal{L}_{UntimeA(A^t_2)} = UntimeL(\mathcal{L}_{A^t_2})$ (2)

Def. 3.6 and (2) imply : $\mathcal{L}_{UntimeA(A^t_1)} = \{TRC \mid \exists\, Trc \in \mathcal{L}_{A^t_1} \text{ with } TRC = UntimeT(Trc)\}$ (3)

$\mathcal{L}_{UntimeA(A^t_2)} = \{TRC \mid \exists\, Trc \in \mathcal{L}_{A^t_2} \text{ with } TRC = UntimeT(Trc)\}$ (4)

Let then $TRC \in \mathcal{L}_{UntimeA(A^t_1)}$:

(3) implies that there exists $Trc \in \mathcal{L}_{A^t_1}$ such that $TRC = UntimeT(Trc)$ (5)

(1) and (5) imply : $Trc \in \mathcal{L}_{A^t_2}$ and $TRC = UntimeT(Trc)$ (6)

(4) and (6) imply : $TRC \in \mathcal{L}_{UntimeA(A^t_2)}$, and therefore $\mathcal{L}_{UntimeA(A^t_1)} \subseteq \mathcal{L}_{UntimeA(A^t_2)}$ ■

5.1.e. Let's prove informally that : $(Trc \in \mathcal{L}_{A^t_2}) \Rightarrow (UntimeT(Proj_{V_1}(Trc)) = Proj_{V_1}(UntimeT(Trc)))$

Def. 3.5 and $Trc \in \mathcal{L}_{A^t_2}$ imply :

Trc and $UntimeT$(Trc) rmodel a same behaviour over alphabet V_2 (1)

$Proj_{V_1}(Trc)$ and $UntimeT(Proj_{V_1}(Trc))$ model a same behaviour over alphabet V_1 (2)

Def. 3.3 and (1) imply : $Proj_{V_1}(Trc)$ and $Proj_{V_1}(UntimeT(Trc))$ model a same behaviour over alphabet V_1 (3)

(2) and (3) imply : $UntimeT(Proj_{V_1}(Trc))$ and $Proj_{V_1}(UntimeT(Trc))$ model a same behaviour over alphabet V_1, and then $UntimeT(Proj_{V_1}(Trc)) = Proj_{V_1}(UntimeT(Trc))$. ■

Proof of Property 6.1.

Let $P1 = (A^{ut} \leq B^{ut})$

$P2 = (\mathcal{L}_{A^{ut}} \subseteq \mathcal{L}_{B^{ut}})$

$P3 = ((TRC \in \mathcal{L}_{A^{ut}}) \Rightarrow (TRC \in \mathcal{L}_{B^{ut}}))$

$P4 = ((TimeT(TRC) \in TimeL(\mathcal{L}_{A^{ut}})) \Rightarrow (TimeT(TRC) \in TimeL(\mathcal{L}_{B^{ut}})))$

$P5 = (TimeL(\mathcal{L}_{A^{ut}}) \subseteq TimeL(\mathcal{L}_{B^{ut}}))$

$P6 = (\mathcal{L}_{Temp(A^{ut})} \subseteq \mathcal{L}_{Temp(B^{ut})})$

$P7 = (Temp(A^{ut}) \leq Temp(B^{ut}))$

By definition : $P1 \Leftrightarrow P2 \Leftrightarrow P3$

Def. 3.6 implies : $P3 \Leftrightarrow P4$

By definition : $P4 \Leftrightarrow P5$

Def. 6.5 implies : $P5 \Leftrightarrow P6$

By definition : $P6 \Leftrightarrow P7$

Therefore : $P1 \Leftrightarrow P7$. ■

Proof of Lemma 6.1.

Let $A^{ut}=UntimeA(A^t)=(Q^{ut},V\cup\{tick\},\delta^{ut},q_0)$ be an untimed automaton, and $A^{tp}=Temp(A^{ut})$.
Since A^{ut} has $|Q^{ut}|$ states, then each of these states can be identifed by $\log_2(|Q^{ut}|)$ bits.
Since there are $|V|$ events, then each event of V can be identified by $\log_2(|V|)$ bits.
Let's compute the following functions :

$\alpha : Q^{ut} \rightarrow Q^{ut}$, where : $([q_1,tick,q_2] \in \delta^{ut} \text{ and } q_1\neq q_2) \Rightarrow (q_2=\alpha(q_1))$
$([q_1,tick,q_1] \in \delta^{ut}) \Rightarrow (\alpha(q_1)=\text{"not defined"})$
$([q_1,tick,q_2] \notin \delta^{ut}) \Rightarrow (\alpha(q_1)=\text{"not defined"})$

Therefore 1 bit is necessary for the value "not defined", and $\log_2(|Q^{ut}|)$ bits are used to define each state.
$\alpha(q)$ are initialized to "not defined" for all states of Q^{ut} : the complexity is in $O(|Q^{ut}|*\log_2(|Q^{ut}|))$
$\alpha(q)$ are computed by going through all transitions of δ^{ut} : the complexity is in $O(|\delta^{ut}|*\log_2(|Q^{ut}|))$
Therefore, the computation of α is in : $O((|Q^{ut}|+|\delta^{ut}|)*\log_2(|Q^{ut}|))$

Let R^{ut} be the set of states q such that $\alpha(q)$ is not defined.
The complexity for computing R^{ut} is in $O(|Q^{ut}|*(\log_2(|Q^{ut}|))$

$\beta : Q^{ut} \rightarrow R^{ut}$, where :
$(q_2=\beta(q_1)) \Leftrightarrow (q_1 \in R^{ut} \Rightarrow q_2=q_1)$
$(q_1 \notin R^{ut} \Rightarrow \exists\, r_0, r_2, ..., r_{k+1} \in Q^{ut}$, with : $r_0=q_1, r_{k+1}=q_2, \alpha(r_i)=r_{i+1}$, for $i=1, ..., k)$
The computation of β (when α is already computed) is in : $O(|Q^{ut}|^2*\log_2(|Q^{ut}|))$

$\gamma : Q^{ut} \rightarrow \{0,1\}$ where : $(\gamma(q_2)=1) \Leftrightarrow (\exists\, q_1$ such that : $q_2=\alpha(q_1))$
$\gamma(q)$ are initialized to 0 for all states of Q^{ut} : the complexity is in $O(|Q^{ut}|*\log_2(|Q^{ut}|))$
$\gamma(q)$ are computed by going through all transitions of δ^{ut} : the complexity is in $O(|\delta^{ut}|*\log_2(|Q^{ut}|))$
Therefore the computation of γ is in : $O((|Q^{ut}|+|\delta^{ut}|)*\log_2(|Q^{ut}|))$

$\mu : Q^{ut}\times V \rightarrow Q^{ut}$, where : $([q_1,\sigma,q_2] \in \delta^{ut}) \Rightarrow (q_2=\mu(q_1,\sigma))$
$([q_1,\sigma,q_2] \notin \delta^{ut}) \Rightarrow (\mu(q_1,\sigma)=\text{"not defined"})$
$\mu(q,\sigma)$ are initialized to "not defined" for all states of Q^{ut} and all events of V:
the complexity is in $O(|Q^{ut}|*|V|*\log_2(|Q^{ut}|*|V|))$
$\mu(q,\sigma)$ are computed by going through all transitions of δ^{ut}: the complexity is in
$O(|\delta^{ut}|*\log_2(|Q^{ut}|*|V|))$
Since $|\delta^{ut}|\leq|Q^{ut}|*|V|$ (A^{ut} is deterministic), the computation of μ is in : $O(|Q^{ut}|*|V|*\log_2(|Q^{ut}|*|V|))$

Step 1: Defining states of A^{tp}
Let $\mathcal{S}_0, \mathcal{S}_1, ..., \mathcal{S}_{n-1}$ be the sets of states of A^{ut} which are closed under tick, and where $n=|R^{ut}|$.
$\mathcal{S}_0$ is the set which contains the initial state q_0 of Q^{ut}.
Formally, two any states q_1 and q_2 of a same $\mathcal{S}_i$ are such that $\beta(q_1)=\beta(q_2)$.
Computation of the sets $\mathcal{S}_i$, for $i=1, ..., |R^{ut}|$:
- Each $\mathcal{S}_i$ contains initially one state -noted r_i- of R^{ut}, and let then the function ν, such that $\nu(r_i)=\mathcal{S}_i$.
  The complexity to initialize all $\mathcal{S}_i$ and to compute ν is in $O(|Q^{ut}|*\log_2(|Q^{ut}|))$.
- Each $\mathcal{S}_i$ must contain all states of Q^{ut} such that $\nu(\beta(q))=\mathcal{S}_i$.
  The complexity to compute all the $\mathcal{S}_i$ is in $O(|Q^{ut}|*\log_2(|Q^{ut}|))$.

All states of a same $\mathcal{S}_i$ are then associated to a same state -identified by r_i- of A^{tp}.

Step 2: Relabeling the states of A^{ut}

For each $\mathcal{S}_i$: (associating an initial n_i-uplet)

- let $q_{i,j}$, for $j=1,..., n_i$, be the states of $\mathcal{S}_i$ such that $\gamma(q)=0$, where n_i is the number of states.
- to each state $q_{i,j}$, we associate the n_i-uplet $(t_1, ..., t_{n_i})$ with $t_j = 0$, and $t_k = \lambda$ if $k \neq j$;
- to all other states of $\mathcal{S}_i$, we associate the initial n_i-uplet $(\lambda, ..., \lambda)$;

  Computing n_i and the states $q_{i,j}$ is in $O(|\mathcal{S}_i|*\log_2(|Q^{ut}|))$

  Associating an initial n_i-uplet to one state of $\mathcal{S}_i$ is in $O(n_i*\log_2(|Q^{ut}|))=O(|\mathcal{S}_i|*\log_2(|Q^{ut}|))$

  Therefore, associating an initial ni-uplet to all states of state of $\mathcal{S}_i$ is in $O(|\mathcal{S}_i|^2*\log_2(|Q^{ut}|))$.

  The complexity for all sets $\mathcal{S}_i$ is then in $O(|Q^{ut}|^2*\log_2(|Q^{ut}|))$.

For each $\mathcal{S}_i$: (associating an n_i-uplet)

- Let e and f be two different states of $\mathcal{S}_i$, respectively associated to the n_i-uplets $(t_1,...,t_{n_i})$ and $(u_1,...,u_{n_i})$. If f is reached from e after a tick then : $u_j=t_j+1$ if $t_j \neq \lambda$, for $j=1, ..., n_i$.

  The biggest length of a sequence of ticks in $\mathcal{S}_i$ is smaller than or equal to $|\mathcal{S}_i|$.

  The number of sequences in $\mathcal{S}_i$ is is smaller than or equal to $|\mathcal{S}_i|$.

  Incrementing one component of an n_i-uplet is in $O(\log_2(|Q^{ut}|))$.

  Therefore, associating an n_i-uplet to all states of state of $\mathcal{S}_i$ is in $O(|\mathcal{S}_i|^2*\log_2(|Q^{ut}|))$.

  Each state of $\mathcal{S}_i$ is then relabeled by $\langle r_i,t\rangle$ where t is an n_i-uplet;

  The complexity for all sets $\mathcal{S}_i$ is then in $O(|Q^{ut}|^2*\log_2(|Q^{ut}|))$.

Step 3 Computing transitions of A^{tp}

$A^{tp} =(Q, V, \delta^{tp}, \mathcal{T}, \mathcal{I} , r_0, t_0,i_0)$

Q is the set of r_i (Step 1);

V is such that $V\cup\{tick\}$ is the alphabet of A^{ut};

The initial state of A^{ut} is $\langle r_0,t\rangle$, where $t=(t_1,...,t_{n_0})$.

i_0 is the smaller index such that $t_{i_0} \neq \lambda$, and $t_0=t_{i_0}$;

Computations of r_0, i_0 and t_0 are in $O(|Q^{ut}|*\log_2(|Q^{ut}|))$

$\mathcal{I} = \{1, 2, ...,\sup(n_i)\} \subseteq \{1, 2, ..., |Q^{ut}|\}$, where $\sup(n_i)$ is the biggest n_i , for $i=1, ..., |R^{ut}|$, (Steps 1 and 2)

$\mathcal{T} \subseteq \{0,1, \sup(|\mathcal{S}_i|)\} \subseteq \{0, 1, ..., |Q^{ut}|\}$

Computations of $\mathcal{I}$ and $\mathcal{T}$ are in $O(|Q^{ut}|*\log_2(|Q^{ut}|))$.

Computation of δ^{tp} :

- Initially, a transition $[\langle r_i,t1\rangle;\sigma;\langle r_j,t2\rangle]$ in A^{ut} implies a transition $[r_i;\sigma;r_j]$;

  For all the transitions of A^{ut}, the complexity is in $O(|\delta^{ut}|*\log_2(|Q^{ut}|^2*|V|))$.
- To all transitions of δ^{tp}, an initial transformation function A is associated, such that A(i,t)="not defined" for any $i\in\mathcal{I}$ and $t\in\mathcal{T}$. The complexity is in $O(|\delta^{ut}|*|Q^{ut}|^2*\log_2(|Q^{ut}|^2))$.
- By going through all transitions of δ^{ut}, the transformation functions of transitions are computed.

  The complexity is in $O(|\delta^{ut}|*|Q^{ut}|^2*\log_2(|Q^{ut}|^2))$..

If we add and simplify all the complexity, we obtain a total complexity in

$$O(|Q^{ut}|*|V|*\log_2(|Q^{ut}|*|V|) + |\delta^{ut}|*|Q^{ut}|^2 *\log_2(|Q^{ut}|))$$

Proof of Theorem 7.1.

After the first step of *Der_Seq_Prot* , the operator *Transf* respects the ordering and the timing requirements between events of SS^t. In fact, $\mathcal{L}_{TSS^t} \subseteq \mathrm{Ext}_{V \cup I}(\mathcal{L}_{SS^t})$ and $\mathrm{Proj}_V(\mathcal{L}_{TSS^t}) = \mathcal{L}_{SS^t}$, where I contains internal events i(q).

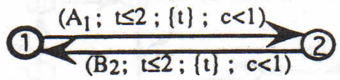
After the second and third steps of *Der_Seq_Prot*, some transitions i(q), without any timing requirements, are replaced by transitions $(s_i^j(q), \mathrm{True}, \{t_{i,j}\})$ and $(r_j^i(q), E_{i,j}(t_{i,j}), \emptyset)$, which contain timing constraints. Therefore, we deduce that :

	$\mathrm{Proj}_V(\mathcal{L}_{MedSS^t}) \subseteq \mathrm{Proj}_V(\mathcal{L}_{TSS^t}) = \mathcal{L}_{SS^t}$	(1)
(1) and Property 5.1.d imply	$UntimeL(\mathrm{Proj}_V(\mathcal{L}_{MedSS^t})) \subseteq \mathcal{L}_{SS^{ut}}$	(2)
Def. 7.3 implies	$PrSS^{ut} = \mathrm{Proj}_V(MedSS^{ut}) = \mathrm{Proj}_V(UntimeA(MedSS^t))$	(3)
(3) is equivalent to	$\mathcal{L}_{PrSS^{ut}} = \mathrm{Proj}_V(UntimeL(\mathcal{L}_{MedSS^t}))$	(4)
(4) and Property 5.1.e imply	$\mathcal{L}_{PrSS^{ut}} = UntimeL(\mathrm{Proj}_V(\mathcal{L}_{MedSS^t}))$	(5)
(2) and (5) imply	$\mathcal{L}_{PrSS^{ut}} \subseteq \mathcal{L}_{SS^{ut}}$, i.e., $PrSS^{ut} \leq SS^{ut}$.	

■

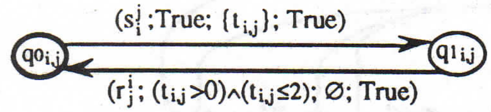
APPENDIX B : Example of Protocol Synthesis

Entries of the procedure *Der_Seq_Prot* **:**



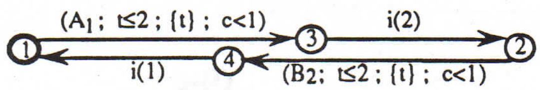
Desired Service SS^t

(Fig. 3.b)



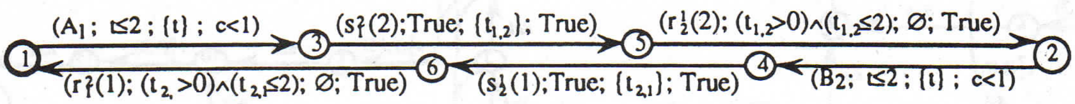
Supremal behaviour of the medium $SupMed^t_{i,j}$, with i, j =1,2 and i≠j (Fig. 4)

Step 1 :



TSS^t=*Transf* (SS^t).

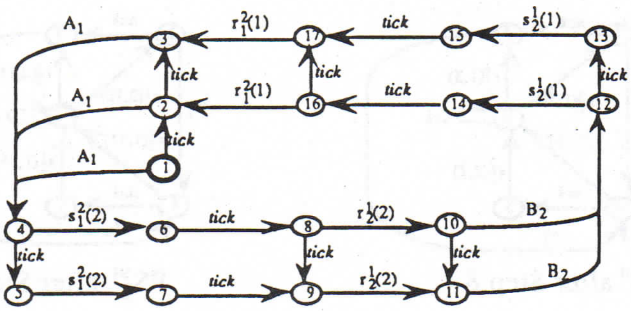
Step 2 :



$MedSS^t_\varepsilon$

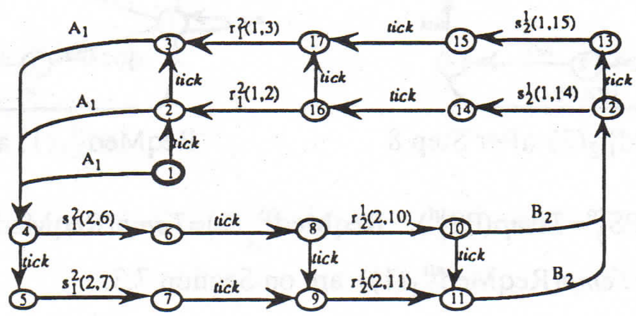
Step 3 : $MedSS^t = MedSS^t_\varepsilon$ because there is no transition ε.

Step 4 :



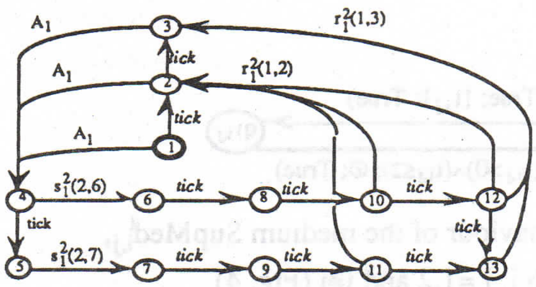
$MedSS^{ut}$ =*UntimeA*($MedSS^t$).

Step 5 :

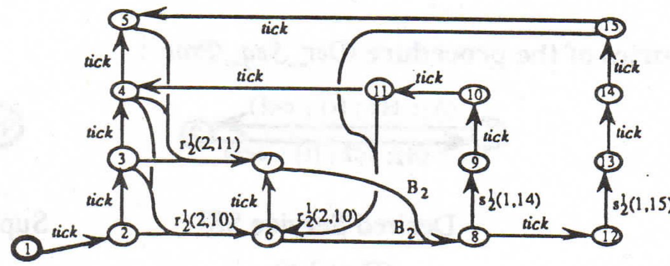


GPS^{ut}.

Step 6 :

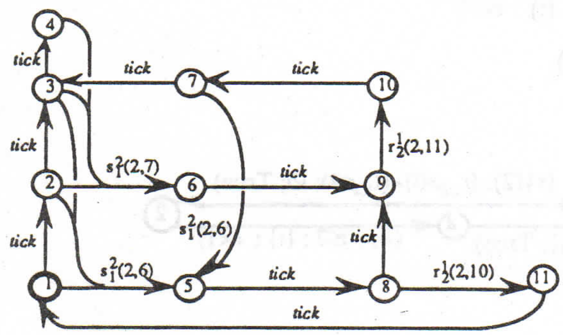


PS_1^{ut}

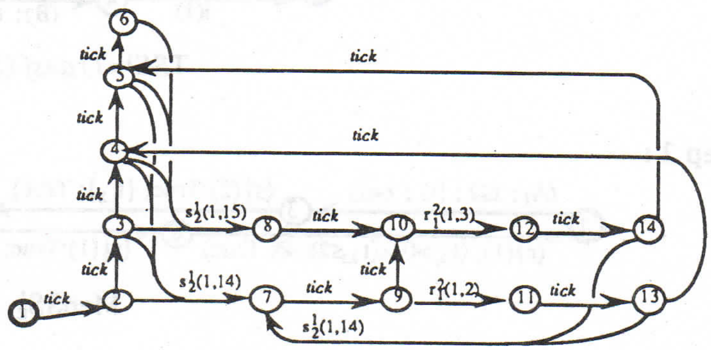


PS_2^{ut}

Step 7 :

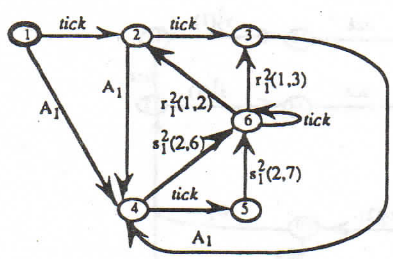


$ReqMed_{1,2}^{ut}(2)$

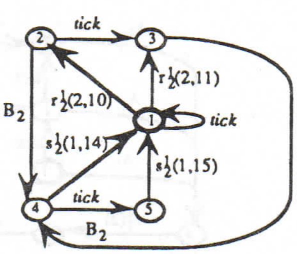


$ReqMed_{2,1}^{ut}(1)$

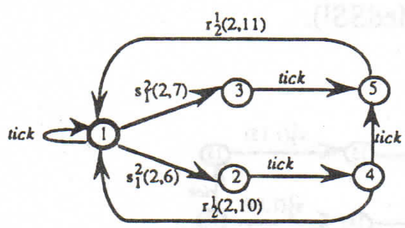
Step 8 :



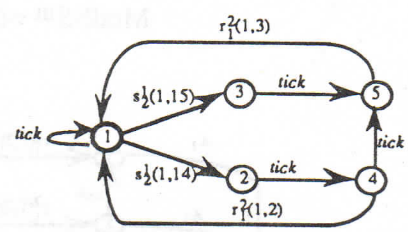
PS_1^{ut} after Step 8



PS_2^{ut} after Step 8



$ReqMed_{1,2}^{ut}(2)$ after Step 8



$ReqMed_{2,1}^{ut}(1)$ after Step 8

Step 9 : PS_1^{tp}=*Temp*(PS_1^{ut}), PS_2^{tp}=*Temp*(PS_2^{ut}), $ReqMed_{1,2}^{tp}(2)$=*Temp*($ReqMed_{1,2}^{ut}(2)$), and $ReqMed_{2,1}^{tp}(1)$=*Temp*($ReqMed_{2,1}^{ut}(1)$) are on Section 7.3.

Step 10 : Results on Section 7.3.